# Looping (II)

# Outline

- Motivation

1. `for` statement

2. Further Examples on Array Processing

3. More For-Loop Examples

# Motivation

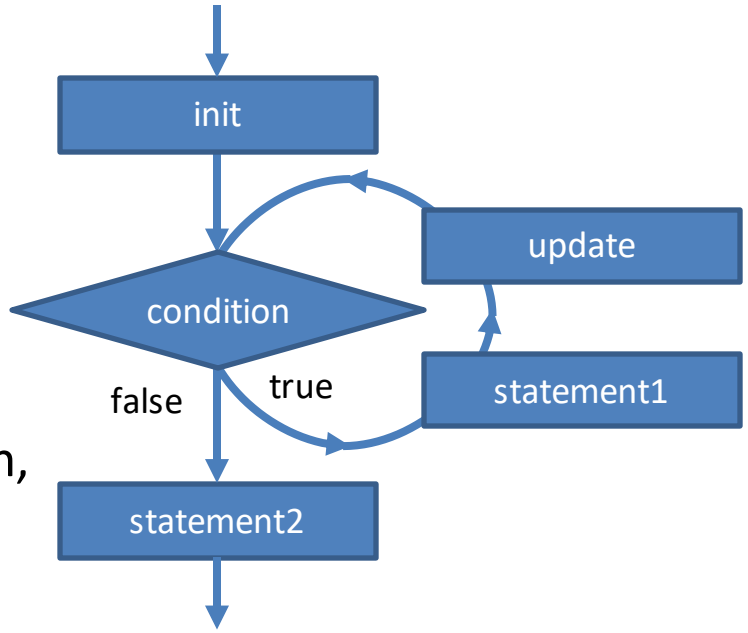
- In our previous lectures and lab, we have learned how to use **while** loop
- We have also seen that there are two <u>Basic Cases</u> of looping:
  - <u>Finite repetition</u>
  - <u>Indefinite repetition</u>
- Today we are going to look at `for` loop, a feature that generally help us do <u>finite repetition</u>

# 1. for statement (*Syntax*)

- The **initialization** (init) statement
  - Executes once before the **condition** statement.

```
for (init; condition; update)
    statement1;
statement2;
```

- The **condition** statement
  - is the same as in the while-loop condition.
- The loop body (**statement1**)
  - Repeats until the **condition** statement becomes ***false***.
- The **update** statement
  - Executes after **statement1** in each iteration, and
  - is usually for updating the loop **condition**.



# 1.1. for Statement (Example #1)

```
1  int i;
2
3  // A simple loop that iterates 5 times
4  for (i = 1; i <= 5; i++) {
5      printf("%d\n", i);
6  }
7
8  printf("Lastly, i = %d\n", i);
9
```

```
1
2
3
4
5
Lastly, i = 6
```

# 1.1. for-loop vs. while-loop

**While-Loop Version**

```
int i;

i = 1;
while (i <= 5) {
    printf("%d\n", i);
    i++;
}
```

**For-Loop Version**

```
int i;

for (i = 1; i <= 5; i++) {
    printf("%d\n", i);
}
```

Comparison

# 1.1. `for`-loop vs. `while`-loop

- They are "equivalent" in terms of what you can accomplish with them.
  - Any task you can accomplish with one of these loop structures, you can also accomplish the task with the other loop structure.

- `for`-loop is more expressive for tasks to be repeated a <u>finite number</u> of times, i.e. **<u>finite repetition</u>**.

# 1.2. Example #2 – Multiple ways of writing a for loop

```c
int i, num, N;
scanf("%d", &N);
// i increases from 0 to N-1
for (i = 0; i < N; i++) {
    num = N - i;
    printf("%d\n", num);
}
```

```c
int i, num, N;
scanf("%d", &N);
// i increases from 1 to N
for (i = 1; i <= N; i++) {
    num = N - i + 1;
    printf("%d\n", num);
}
```

```c
int i, num, N;
scanf("%d", &N);
// i decreases from N to 1
for (i = N; i >= 1; i--) {
    num = i;
    printf("%d\n", num);
}
```

Different ways to output numbers from N to 1 using a `for` loop.

The numbers we want to generate inside a loop can usually be expressed in terms of the loop variable.

# 1.3. Example #3: Arrays and for loop

```
int list[4], i;

printf("Enter 4 #'s: ");
for (i = 0; i < 4; i++)
    scanf("%d", &list[i]);

// Print the input values in reverse order
printf("You have entered:");
for (i = 0; i < 4; i++)
    printf("%d", list[ 4 - i - 1 ]);
printf("\n");
```

We usually use for loops to process array elements.

```
Enter 4 #'s: 7 11 45 23
You have entered: 23 45 11 7
```

Note: Input values can be separated by any white space character.

# 2. Further Examples on Array Processing

- The following few slides are some examples illustrating how we can process data in arrays using `for` loops. The examples include:
  - Reading *N* numbers from the user and store the numbers in an array
  - Computing *average* of all the numbers in an array
  - Finding the largest value in an array

```c
int N,              // # of data
    num[100],       // To store up to 100 numbers
    k;

printf("# of data:");
scanf("%d", &N);    // Assuming 1 <= N <= 100

for ( k = 0; k < N; k++ )
   scanf("%d", &num[ k ]);

```

**Example 4:** Read N integers (N <= 100) from the user and store the integers in the array `num[ ]`.

```c
// Suppose num[] contains N numbers.
// (Please refer to Example 4 for the code)

int i;
double sum;

sum = 0.0;
for ( i = 0; i < N; i++ )
   sum += num[ i ];

printf("Average = %.2f\n", sum / N);
```

**Example 5.** Computing the average of the numbers in an array.

```c
// Suppose num[] contains N numbers.
// (Please refer to Example 4 for the code)

int i, max;

max = num[0]; // Assume the 1st element is largest

for ( i = 1; i < N; i++ ) {

    // Update max if we encounter a larger value
    if (num[i] > max)
        max = num[i];
}

printf("The largest # is %d\n", max);
```

**Example 6:** Finding the largest number in an array.

# 2.1. A Common Mistake

- Typical cause of array out-of-bound exception when using arrays in a for-loop:

```
int A[10], i;

for ( i = 0; i <= 10; i++ )
    A[i] = i;
```

Remember, the index of an array element ranges from 0 to “array size - 1”!

# 3. More For-Loop Examples

- For-loops can be used in solving many general problems that may not involve arrays.
- A reminder though: it is usually **NOT** recommended to use them in indefinite repetitions.

# Example #7

- **Objective:** To output the first *N* numbers in the following number series:

  1 2 -3 -4 5 6 -7 -8 9 10 -11 -12 …

Notice that these numbers follow the following pattern:

+ve +ve -ve -ve +ve +ve -ve -ve …

and the pattern "+ve +ve -ve -ve" repeats after every <u>four</u> numbers.

# Example #7: Solution

```
int i, N;

printf("N = ? ");
scanf("%d", &N);

for (i = 1; i <= N; i++) {
    if (i % 4 == 1 || i % 4 == 2)
        printf("%d ", i);
    else
        printf("%d ", -i);
}
```

| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| i % 4 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 |

The remainders, i % 4, also repeat every four numbers

# Summary

- Syntax of `for` loops
- Comparison of for `loops` and `while` loops
- More examples on array processing

# Reading Assignment

- C: How to Program, 8th ed, Deitel and Deitel
- Chapter 3 Structured Program Development in C
  - Sections 3.7 – 3.9
- Chapter 4 C Program Control
  - Sections 4.1 – 4.6, 4.9