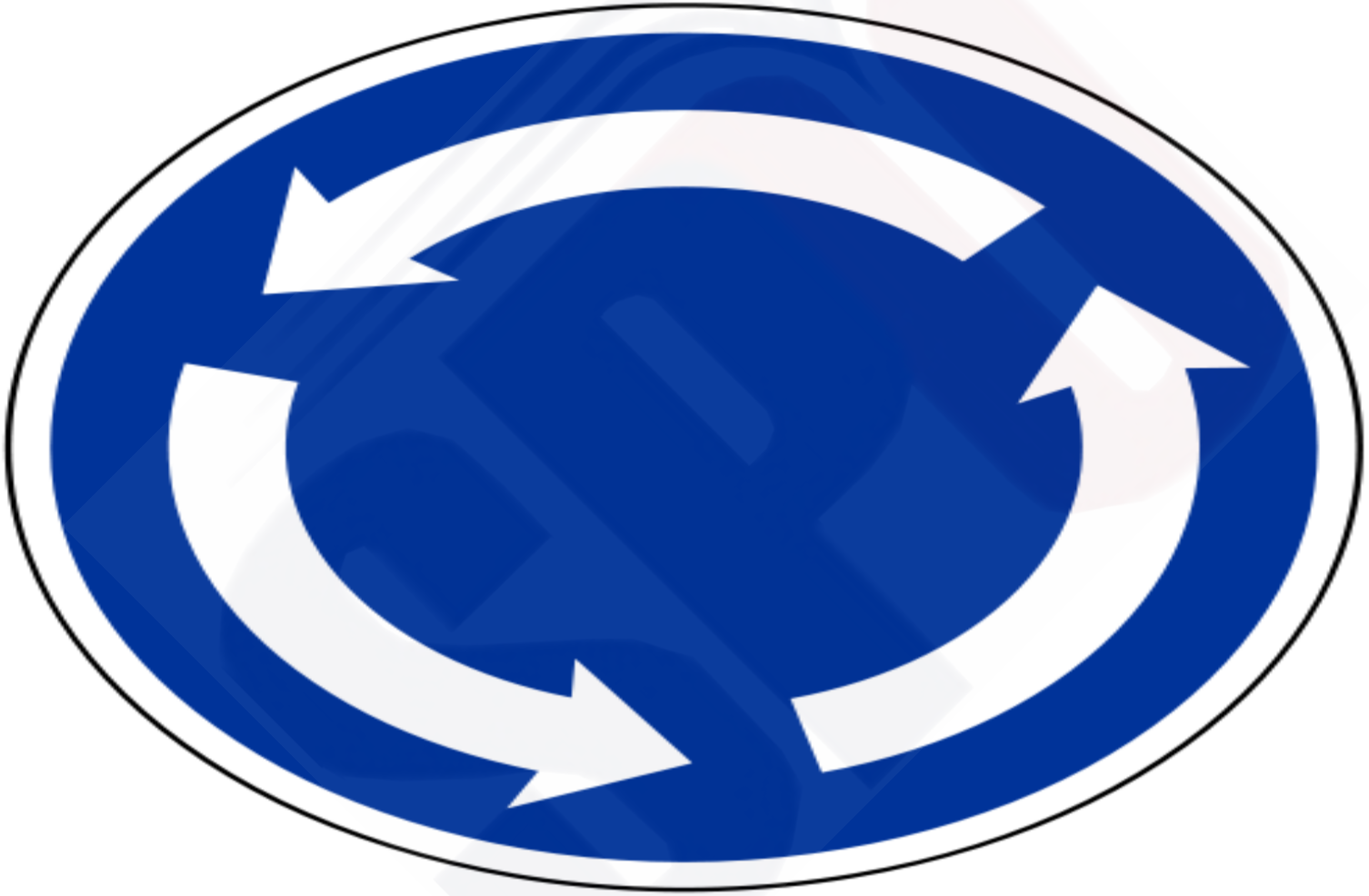
Lecture 5

Loops & Arrays

Dr. Mohammad Ahmad

Loop analogy (roundabout)



Loop

Exiting a Loop

Ninja Cat

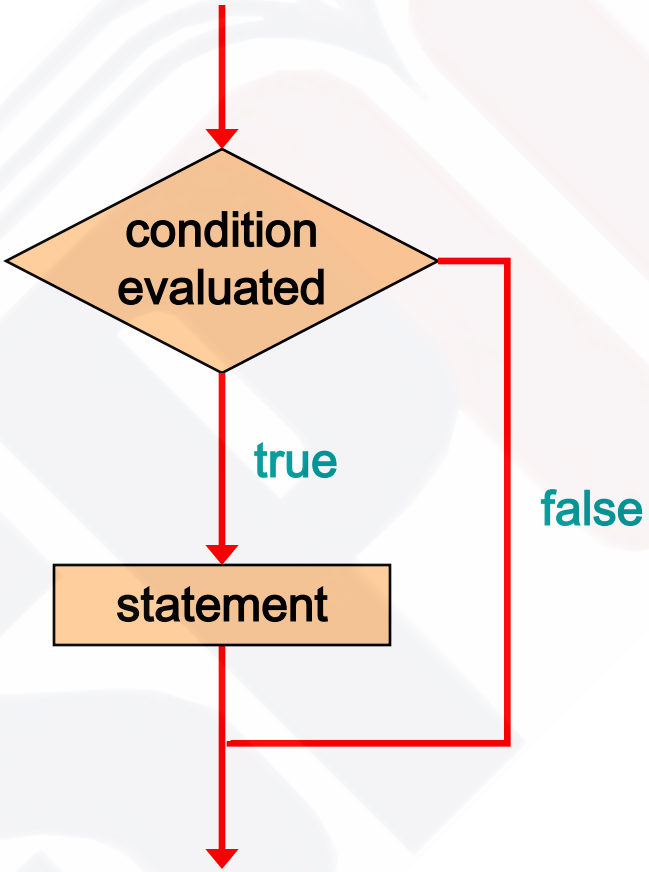
Repetition Statements

- ***Repetition statements*** **allow us to execute a statement multiple times**
- **Often they are referred to as *loops***
- **C has three kinds of repetition statements:**
  - **the *while loop***
  - **the *do loop***
  - **the *for loop***
- **The programmer should choose the right kind of loop for the situation**

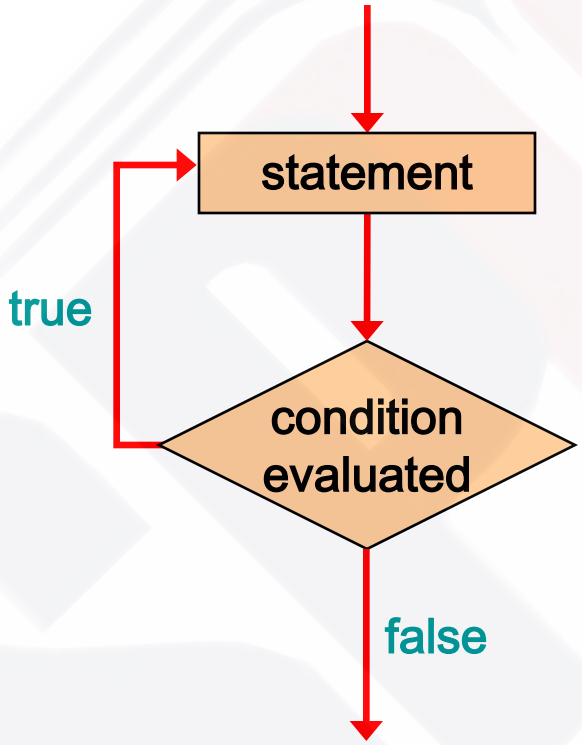
There are three loop constructs in C

- do-while loop (or do loop for short)
- while loop
- for loop
- Loops = repetition statements

Logic of an if statement



Logic of a do Loop



The do Statement

- A *do statement* has the following syntax:

```
do
{
    statement;
}
while ( condition );
```

  - The `statement` is executed once initially, and then the `condition` is evaluated
  - The statement is executed repeatedly until the condition becomes false

The do Statement

- **An example of a do loop:**

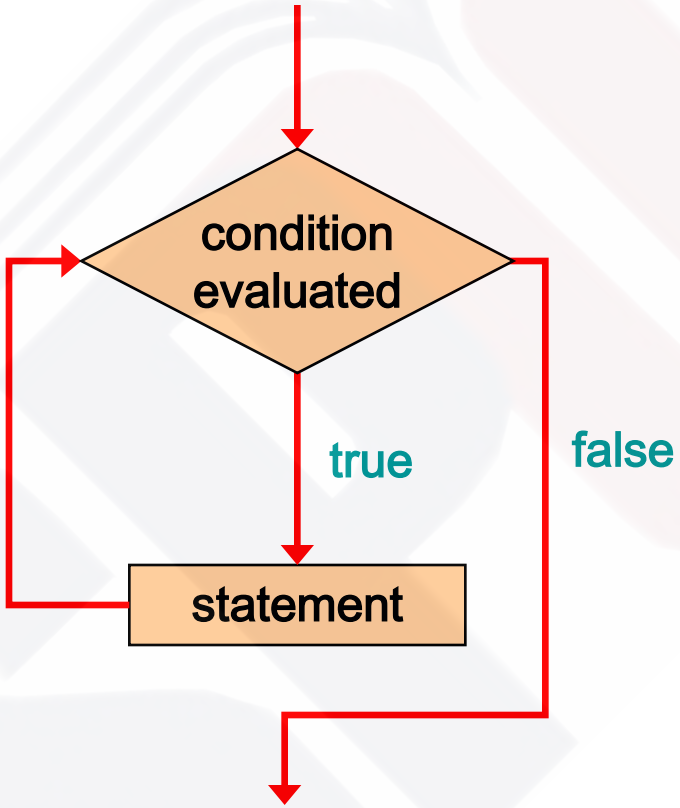
```
int count = 0;
do
{
   count++;
   printf(“%d\n”, count);
} while (count < 5);
```

- **The body of a do loop is executed at least once**

Example: Fixing Bad Keyboard Input

- **Write a program that refuses to accept a negative number as an input.**
- **The program must keep asking the user to enter a value until he/she enters a positive number.**
- **How can we do this?**

Logic of a while Loop



The while Statement

- A *while statement* has the following syntax:

```
while ( condition )
    statement;
```

  - If the `condition` is true, the `statement` is executed
  - Then the condition is evaluated again, and if it is still true, the statement is executed again
  - The statement is executed repeatedly until the condition becomes false

The while Statement

- **An example of a while statement:**

```
int count = 1;
while (count <= 5)
{
   printf (“%d\n”, count);
   count++;
}
```

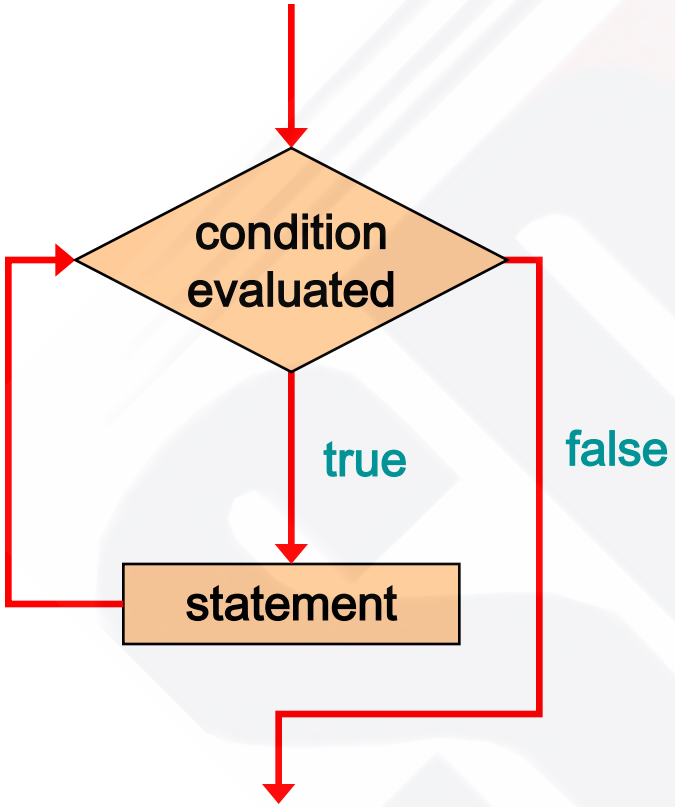
  - **If the condition of a `while` loop is false initially, the statement is never executed**
  - **Therefore, the body of a `while` loop will execute zero or more times**

The while Statement

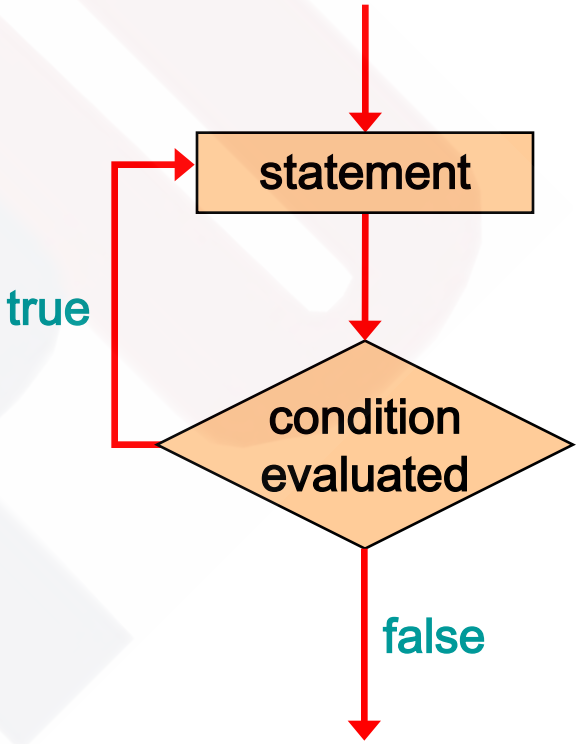
- **Let's look at some examples of loop processing**
- **A loop can be used to maintain a *running sum***
- **A *sentinel value* is a special input value that represents the end of input**
- **A loop can also be used for *input validation*, making a program more *robust***

Comparing while and do

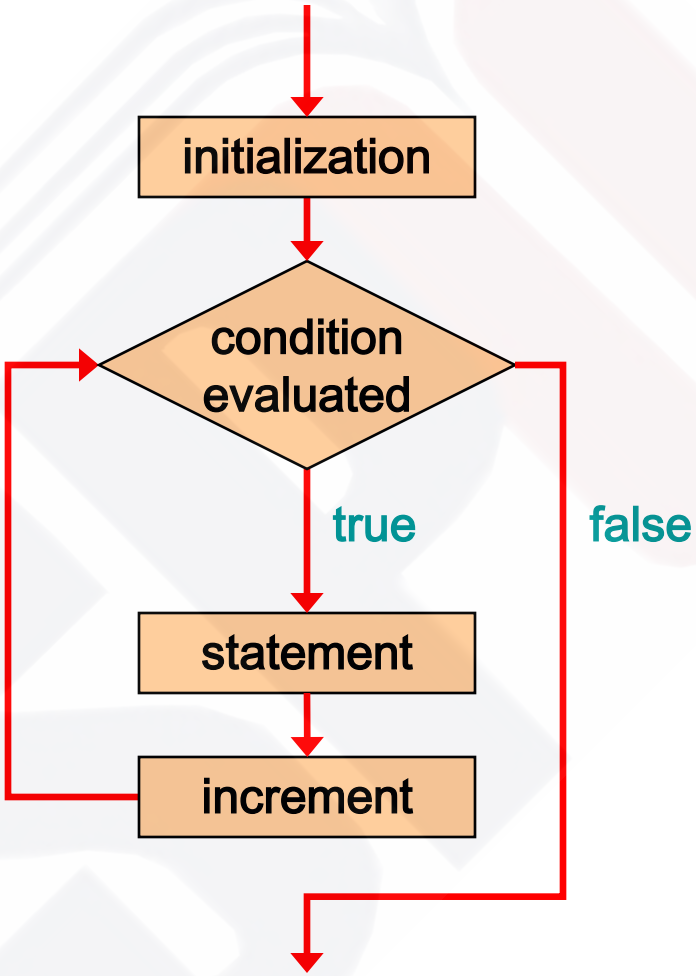
The while Loop



The do Loop



Logic of a for loop



The for Statement

- **A *for statement* has the following syntax:**

The *initialization* is executed once before the loop begins

The *statement* is executed until the *condition* becomes false

```
for ( initialization ; condition ; increment )
   statement;
```

The *increment* portion is executed at the end of each iteration

The for Statement

- **A `for` loop is functionally equivalent to the following `while` loop structure:**

```
initialization;
while ( condition )
{
    statement;
    increment;
}
```

The for Statement

- **An example of a `for` loop:**

```
for (int count=1; count <= 5; count++)
   printf (“%d\n”, count);
```

  - **The initialization section can be used to declare a variable**
  - **Like a `while` loop, the condition of a `for` loop is tested prior to executing the loop body**
  - **Therefore, the body of a `for` loop will execute zero or more times**

The for Statement

- **The increment section can perform any calculation**

```
Int num;
for (num=100; num > 0; num -= 5)
   printf (“%d\n”, num);
```

- **A `for` loop is well suited for executing statements a specific number of times that can be calculated or determined in advance**

The for Statement

- Each expression in the header of a `for` loop is optional
- If the initialization is left out, no initialization is performed
- If the condition is left out, it is always considered to be true, and therefore creates an infinite loop
- If the increment is left out, no increment operation is performed

Infinite Loops

- **The body of a `while` loop eventually must make the condition false**
- **If not, it is called an *infinite loop*, which will execute until the user interrupts the program**
- **This is a common logical error**
- **You should always double check the logic of a program to ensure that your loops will terminate normally**

Infinite Loops

- **An example of an infinite loop:**

```
int count = 1;
while (count <= 25)
{
    printf (“%d\n”, count);
    count = count - 1;
}
```

  - **This loop will continue executing until interrupted (Control-C) or until an underflow error occurs**

Nested Loops

- Similar to nested `if` statements, loops can be nested as well
- That is, the body of a loop can contain another loop
- For each iteration of the outer loop, the inner loop iterates completely

Nested Loops

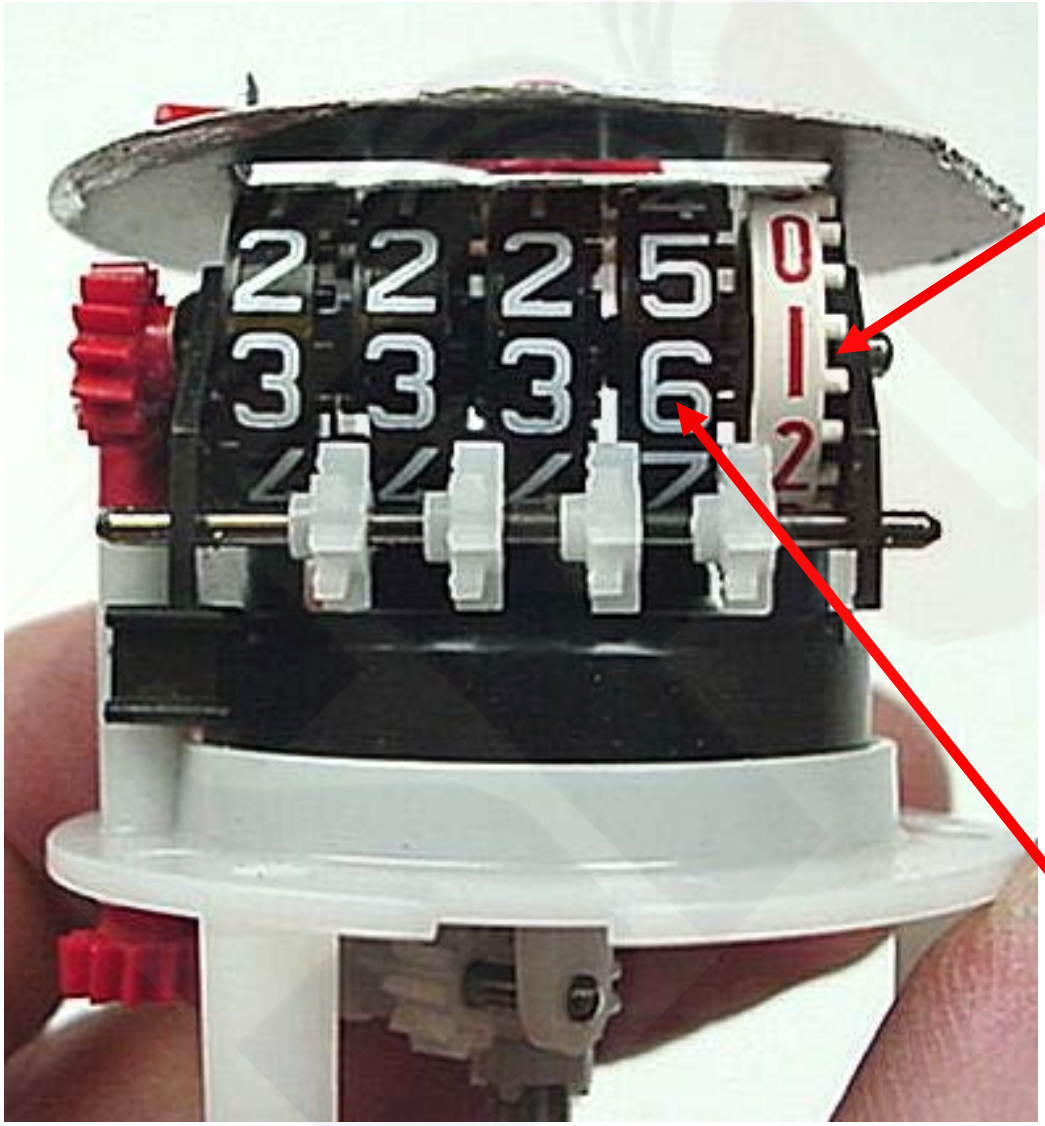
- **How many times will the string "Here" be printed?**

```
count1 = 1;
while (count1 <= 10)
{
   count2 = 1;
   while (count2 <= 20)
   {
      printf ("Here");
      count2++;
   }
   count1++;
}
```

10 * 20 = 200

Analogy for Nested Loops

Analogy for Nested Loops



Inner Loop

Outer Loop

Example: Stars

- **Write a program that prints the following**

```
*
**
***
****
*****
******
*******
********
*********
**********
```

Example: Multiplication Table

Arrays

Problem:
Read 10 numbers from the keyboard and store them

Problem:
Read 10 numbers from the keyboard and store them

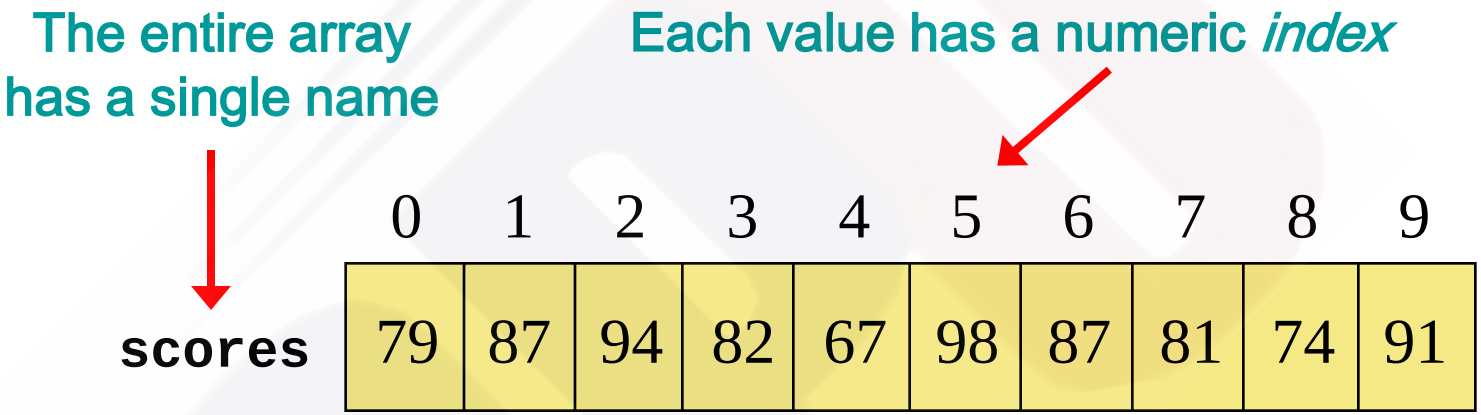
```
// solution #1
int a0, a1, a2, a3, a4, a5, a6, a7, a8, a9;

printf(“Enter a number: “);
scanf(“ %d”, &a0);

printf(“Enter a number: “);
scanf(“ %d”, &a1);

//…

printf(“Enter a number: “);
scanf(“ %d”, &a9);
```

Arrays

- **Arrays are C data types that help us organize large amounts of information**

Arrays

- **An *array* is an ordered list of values**



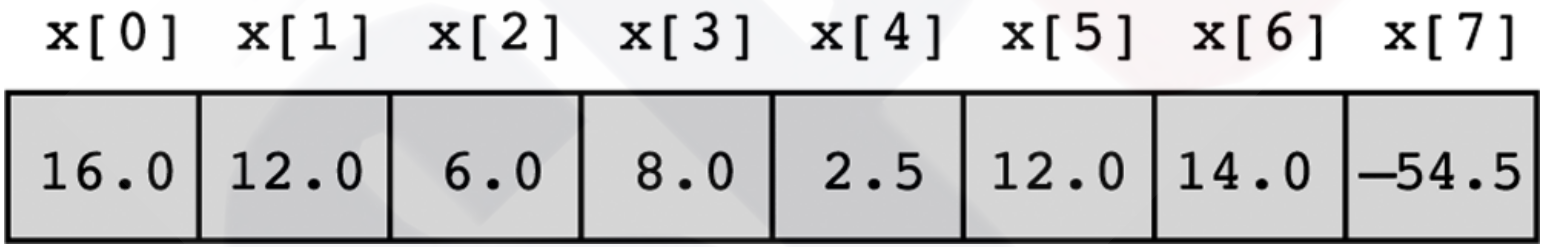
An array of size N is indexed from zero to N-1

This array holds 10 values that are indexed from 0 to 9

An array with 8 elements of type double

```
double x[8];
```

Array x



Problem:
Read 10 numbers from the keyboard and store them

```
// solution #2
int a[10];  // use an array
for(i=0; i< 10; i++)
{
  printf(“Enter a number: “);
  scanf(“ %d”, &a[i]);
}
```

Arrays

- A particular value in an array is referenced using the array name followed by the index in brackets
- For example, the expression

```
scores[2]
```

  refers to the value 94 (the 3rd value in the array)

- That expression represents a place to store a single integer and can be used wherever an integer variable can be used

Arrays

- **For example, an array element can be assigned a value, printed, or used in a calculation:**

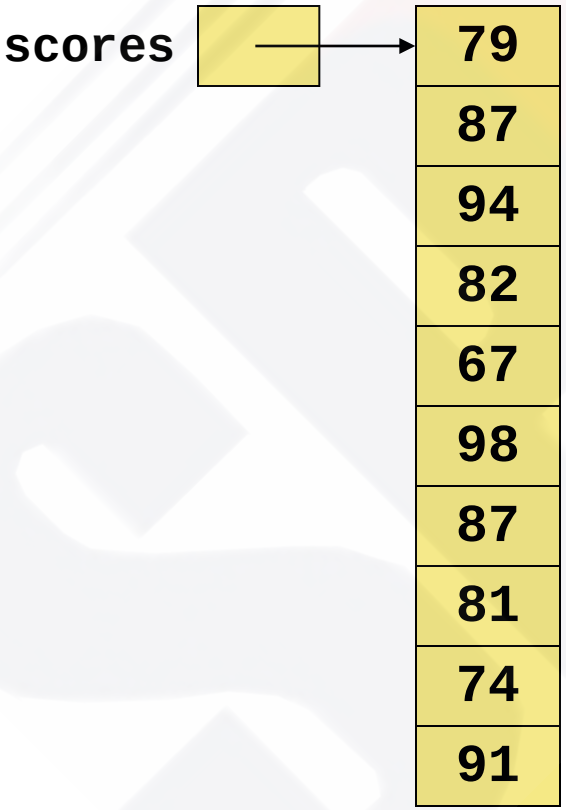
```
scores[2] = 89;

scores[first] = scores[first] + 2;

mean = (scores[0] + scores[1])/2;

printf ("Top = %d“, scores[5]);
```

Arrays

- **The values held in an array are called *array elements***
- **An array stores multiple values of the same type – the *element type***
- **The element type can be a primitive type**
- **Therefore, we can create an array of integers, an array of floats, an array of `doubles`.**

Arrays

- **Another way to depict the `scores` array:**



Declaring Arrays

- **It is possible to initialize an array when it is declared:**

```
float prices[3] = {1.0, 2.1, 2.0};
```

- **Or to initialize it later:**

```
int a[6];

a[0]=3;

a[1]=6;
```

Declaring Arrays

- **Declaring an array of characters of size 3:**

```
char letters[3] = {‘a’, ‘b’, ‘c’};
```

- **Or we can skip the 3 and leave it to the compiler to estimate the size of the array:**

```
char letters[] = {‘a’, ‘b’, ‘c’};
```

For loops and arrays

```
#define N 10
int a[N];
int i;
…
for(i=0; i < N; i++)
  printf(“%d\n”, a[i]);


for(i=0; i <= N; i++)   // this is an error
  printf(“%d\n”, a[i]); // out of bounds
```

For loops and arrays

```
#define N 10
int a[N+1];
int i;
…
for(i=0; i <= N; i++)
  printf(“%d\n”, a[i]);
```

Problem:

Input 10 student IDs and their corresponding grades (A through F). Then find out the number of As, and print the names of the students that got an A.

Comparing Float Values

- You should rarely use the equality operator (==) when comparing two floating point values (`float` or `double`)
- Two floating point values are equal only if their underlying binary representations match exactly
- Computations often result in slight differences that may be irrelevant
- In many situations, you might consider two floating point numbers to be "close enough" even if they aren't exactly equal

Comparing Float Values

- To determine the equality of two floats, you may want to use the following technique:

```
if (fabs(f1 - f2) < TOLERANCE)
   printf ("Essentially equal");
```

  - If the difference between the two floating point values is less than the tolerance, they are considered to be equal
  - The tolerance could be set to any appropriate level, such as 0.000001

Comparing Characters

- As we've discussed, C character data is based on the ASCII character set
- ASCII establishes a particular numeric value for each character, and therefore an ordering
- We can use relational operators on character data based on this ordering
- For example, the character `'+'` is less than the character `'J'` because it comes before it in the ASCII character set