Module 2

Contents of Module 2

- Arithmetic expressions and precedence, Conditional Branching and Loops: Writing and evaluation of conditionals and consequent branching, Iteration and loops
- Arrays: Arrays (1-D, 2-D), Character arrays and Strings

Objectives of Module 2

- Learn arithmetic expressions and precedence.
- Identify need and use of conditional branching.
- Learn loops, their types and syntax.
- Use arrays for storing homogenous data.

Decision Control

- Sometimes situation arises where some decision should be taken based on certain condition
- Ex- if age is >=18, then we can vote
- In C, the following are used for condition checking
  - if
  - if-else
  - switch-case

if statement

- Syntax:

```
if (condition)
{
statements; // if condition is true
}
```

- Ex-

```
if (marks >= 40)
{
printf("You are passed");
}
```

Program using if

```
int marks = 50;
if (marks >= 40)
printf("You have passed");
```

//Here, output will be ‘ You have passed’ because condition is satisfied

if –else statement

- Syntax:

```
if (condition)
{
statements; // if
      condition is true
}
else
{
     statements; // if
      condition is false
}
```

- Example:

```
if (marks >= 40)
{
printf(" You are passed
");
}
 else
 {
  printf(" You are  failed
");
  }
```

Program using if-else

```
int marks = 50;
if (marks >= 40)
printf(“You have passed”);
else
 printf(“You have failed”);
```

Switch Case

- A switch statement tests the value of a variable and compares it with multiple cases
- Once the case match is found, a block of statements associated with that particular case is executed
- The value provided by the user is compared with all the cases inside the switch block until the match is found
- It provides a more efficient and concise way to handle multiple conditional branches compared to using multiple if statements.

Switch Case

- If a case match is not found, then the default statement is executed
- Syntax :

```
switch (n)
        {
        case 1: // code to be executed if n = 1
        break;
        case 2: // code to be executed if n = 2
        break;
        default: // code to be executed if n
                    doesn't match any cases
        }
```

Break

The break statement is used primarily within loop and switch statements. Its purpose is to control the flow of execution by "breaking out" of a loop. Break Statement is a loop control statement that is used to terminate the loop. As soon as the break statement is encountered from within a loop, the loop iterations stop there.

Points to be noted in Switch - Case

- Duplicate case values are not allowed
- The default statement is optional
  - Even if the switch case statement do not have a default statement, it would run without any problem

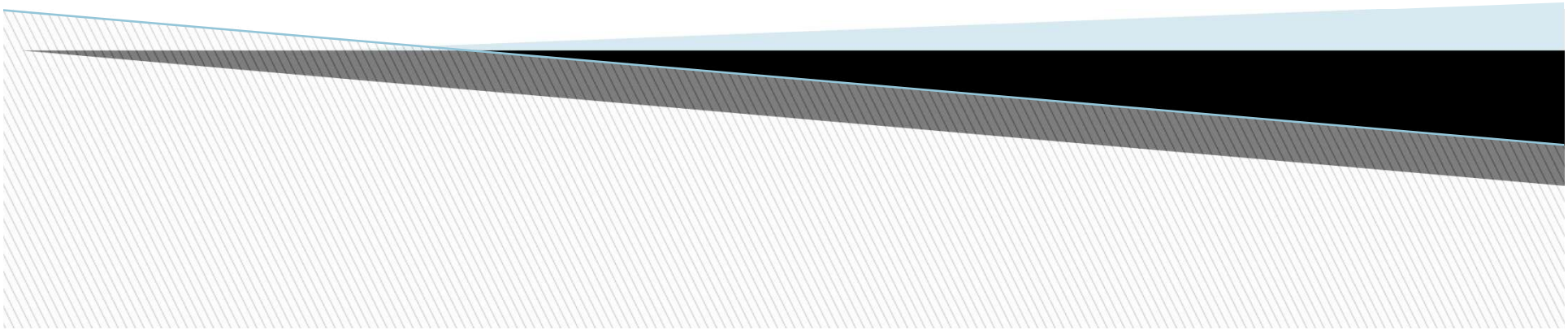
Program Using Switch Case


```
#include <stdio.h>
int main() {
int choice;
printf("Enter a number (1-2): ");
scanf("%d", &choice);
switch (choice) {
case 1:
 printf("You selected option 1.\n");
 break;
    case 2:
          printf("You selected option 2.\n");
          break;
default:
          printf("Invalid choice. Please enter a number between 1 and 2.\n");
          break;
   }

   return 0;
}
```

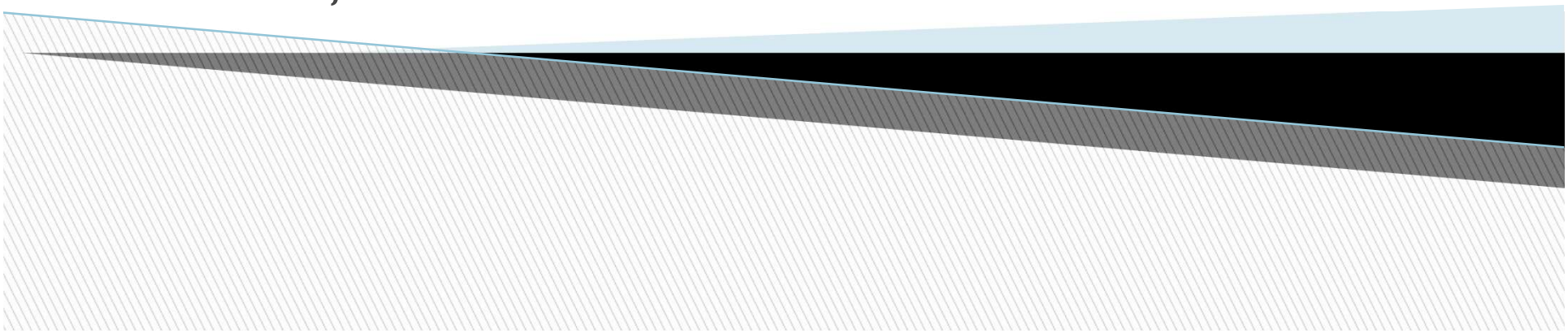
Output

```
Enter a number (1-2): 2
You selected option 2.
Enter a number (1-2): 4
Invalid choice. Please enter a number between 1 and 2.
```



Conditional Branching

Conditional branching in C refers to the ability to execute different code blocks or statements based on certain conditions or expressions. It allows you to control the flow of your program by making decisions at runtime. Conditional branching is typically achieved using conditional statements, such as if, else etc.



Loop

Generally, statements are executed one after other, but (say) if same task needed for certain numbers of times?

- Loops in C is used to execute the block of code several times according to the condition given in the loop.
- Using loop, set of statements are executed repeatedly (iteration)
- Iteration - number of times the body of loop (statements defined in loop) is executed

Types of Loop

- There are three types:
  - while loop
  - for loop
  - do-while loop

While Loop

- A *while* loop in C program repeatedly execute statements defined in the body of loop as long as given condition is true
- Syntax:

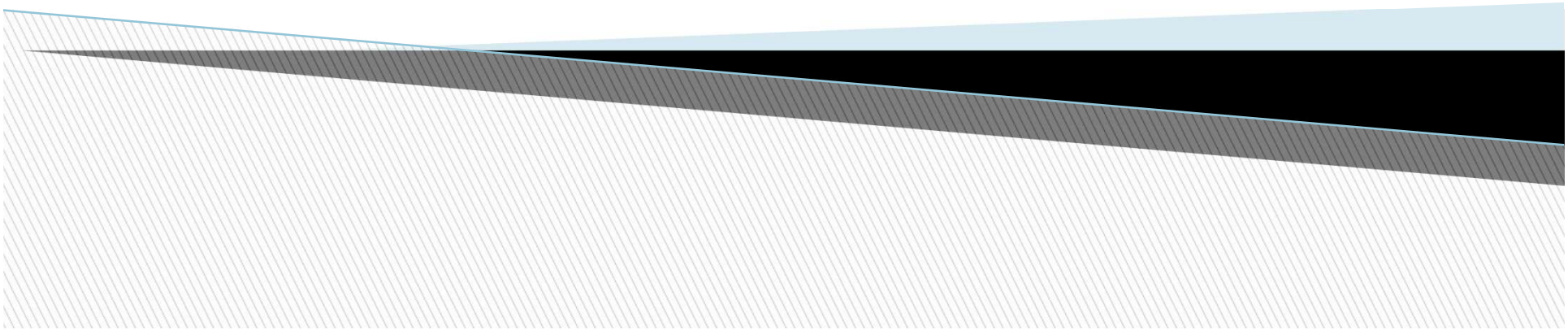
```
while (condition test)
{
    //statement to be executed repeatedly
   //increment (++) Or decrement (--)
}
```

Program using while loop

```
#include <stdio.h>
int main() {
    int count = 1;
 while (count <= 5) {
        printf("This is iteration %d\n", count);
        count++;
    }
return 0;
}
```

Output

This is iteration 1
This is iteration 2
This is iteration 3
This is iteration 4
This is iteration 5



For Loop

- This is used when you know the number of iterations in advance and typically consists of three parts: initialization, condition, and increment (or decrement).
- Syntax:

```
for (initialization; condition; modification)
{
        body of loop
}
```

- here, modification mean increment or decrement

do-while Loop

- There is a minor difference between the working of while and do-while loops
- This difference lie in the place where the the condition is tested
- The *while* loop test the condition **before** execution of any of the statement within the while loop whereas, the do-while test the condition **after** having executed the statement within the loop

Syntax


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

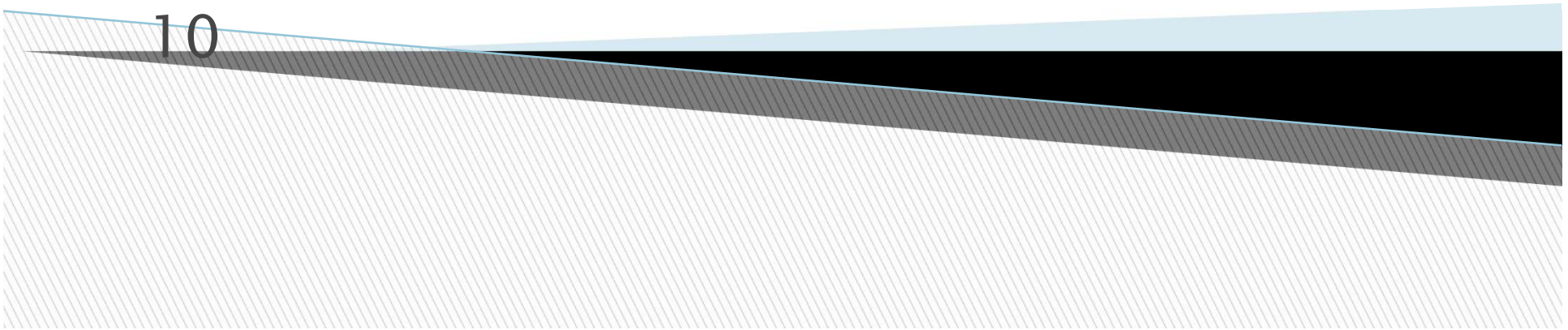
Program using do-while loop

TABLE OF 1

```
#include<stdio.h>
int main(){
int i=1;
do{
printf("%d \n",i);
i++;
}while(i<=10);
return 0;
}
```

Output

1
2
3
4
5
6
7
8
9
10



Comparison Between For Loop, While Loop & Do-While Loop

For loop	While loop	Do while loop
Syntax: For(initialization; condition;updating), { . Statements; }	Syntax: While(condition), { . Statements; . }	Syntax: Do { . Statements; } While(condition);
It is known as entry controlled loop	It is known as entry controlled loop.	It is known as exit controlled loop.
If the condition is not true first time than control will never enter in a loop	If the condition is not true first time than control will never enter in a loop.	Even if the condition is not true for the first time the control will enter in a loop.
There is no semicolon; after the condition in the syntax of the for loop.	There is no semicolon; after the condition in the syntax of the while loop.	There is semicolon; after the condition in the syntax of the do while loop.

Array

- Collection of elements of similar (homogenous) datatypes (int/float/char etc)
- Syntax: data type array_name [size];

Example:

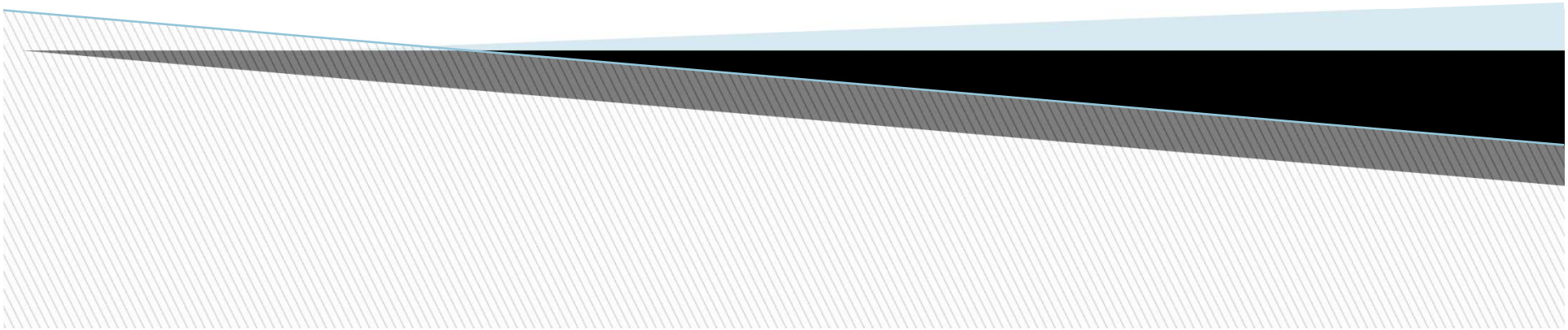
Declaration: To declare an array in C, you specify the data type of its elements and the number of elements it can hold. For example:

int myArray[5]; **// Declares an integer array with 5 elements**

CONTD..

Initialization: One can initialize an array at the time of declaration.

```
int myArray[5] = {1, 2, 3, 4, 5}; // Initializing at declaration
```



CONTD..

Accessing Elements: Array elements are accessed using an **index,** starting from 0 for the first element.

```
int x = myArray[2]; // Accesses the third element (with index 2)
```

- **Index:** An index in an array is a numeric value used to identify and access individual elements within the array.

CONTD..

Zero-Based Indexing: C uses 0 to (n-1) as the array bounds.

Example: int arr[5] = {6,5,1,9,2}; **//'arr' goes from 0 → 4**

Note:

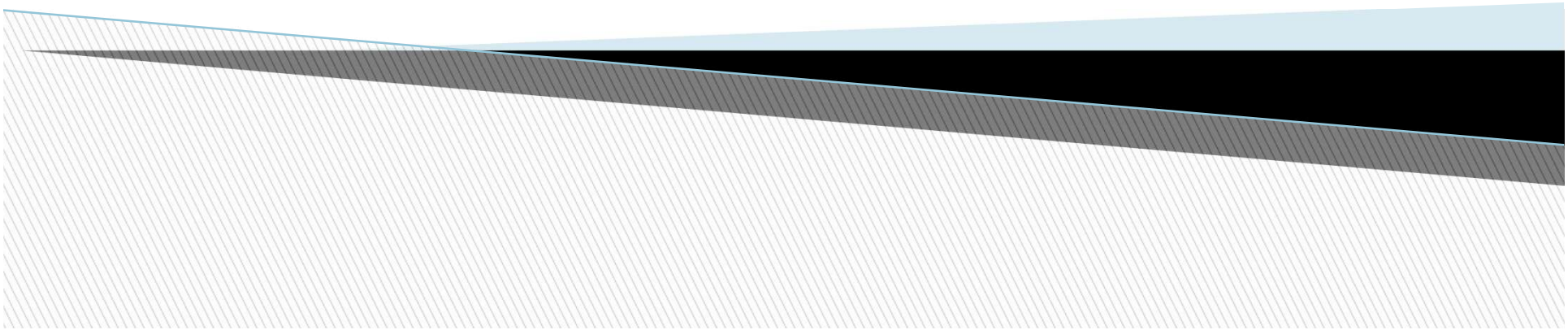
Bounds Checking: Array indexes should be within the valid range of the array. Accessing an element with an index outside this range can lead to errors, such as "index out of bounds" or undefined behavior. It's essential to ensure that your index values are valid.

CONTD..

Size: You can find the number of elements in an array by dividing the total size of the array by the size of one element.

```
int size = sizeof(myArray) / sizeof(myArray[0]);
```

sizeof(myArray): This part of the expression calculates the total size (in bytes) of the entire array.



CONTD..

sizeof(myArray[0]): This part calculates the size (in bytes) of a single element within the array “myArray”. (myArray[0]) represents the first element of the array.

sizeof(myArray) / sizeof(myArray[0]): divides the total size of the array by the size of a single element. This division yields the number of elements in the array.

Array declaration and representation

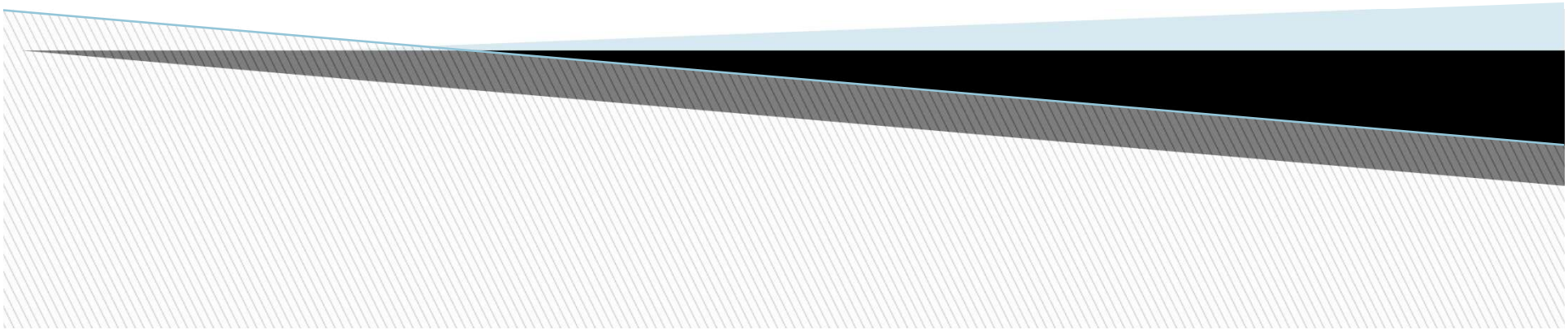
- C uses 0 to (n-1) as the array bounds
  - int arr[5] = {6,5,1,9,2}; **//'arr' goes from 0 → 4**

6	5	1	9	2
arr[0]	**arr[1]**	**arr[2]**	**arr[3]**	**arr[4]**

EXAMPLE

```
int myNumbers[] = {25, 50, 75, 100};
printf("%d", myNumbers[0]);

// Outputs 25
```



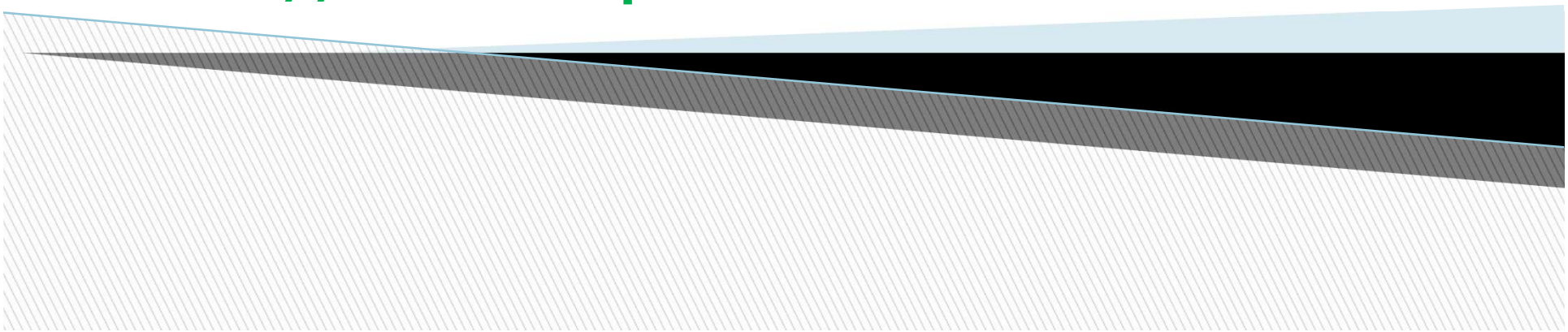
Change an Array Element

In C, one can change an element of an array by directly assigning a new value to the desired element using the array's index.

Example:

```
int myNumbers[4] ={25, 50, 75, 100};
myNumbers[0] = 33;
printf("%d", myNumbers[0]);

// Now output is 33 instead of 25
```



Types of Array

- **One dimensional array:** The One-dimensional arrays, also known as 1-D arrays in C are those arrays that have only one dimension.
- **Example:**

CONTD..

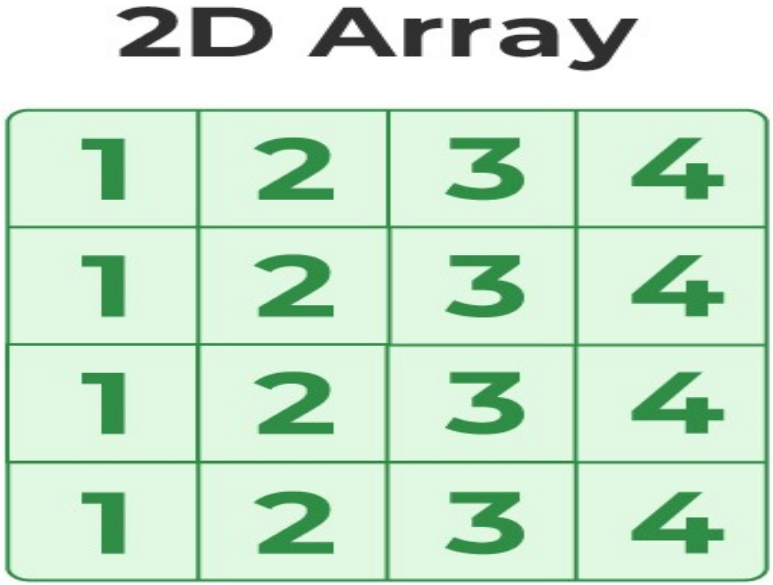
Multidimensional Array: Multi-dimensional Arrays in C are those arrays that have more than one dimension. Generally we study 2D array.

Syntax: data-type array_name[size1] [size2];

size1: Size of the first dimension.(row size)

size2: Size of the second dimension.(column size)

CONTD..

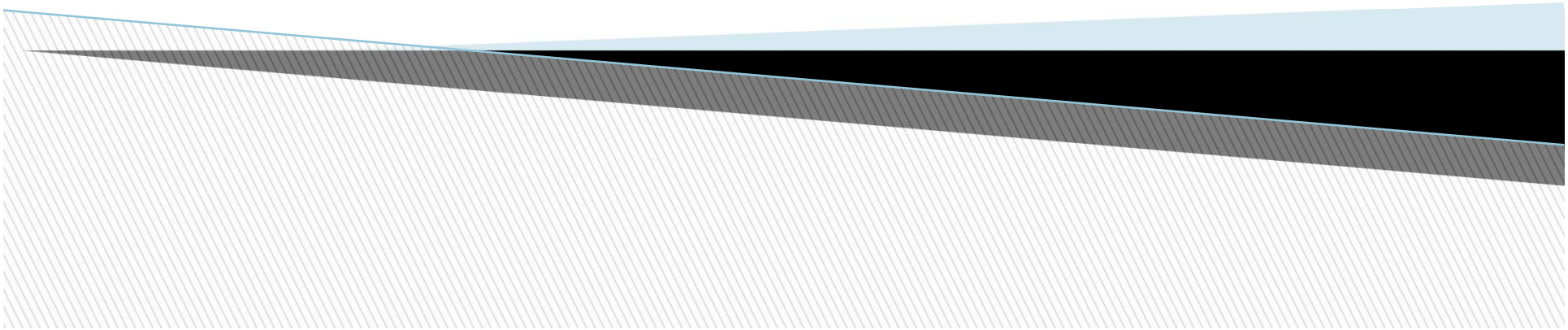


Program using 1-D Array

```
#include <stdio.h>
int main() {
    // Declaration and initialization of a 1D integer array
int numbers[5] = {10, 20, 30, 40, 50};
    // Accessing and printing elements of the array
printf("Elements of the array:\n");
 for (int i = 0; i < 5; i++) {
        printf("numbers[%d] = %d\n", i, numbers[i]);
    }
return 0;
}
```

Output

Elements of the array:
numbers[0] = 10
numbers[1] = 20
numbers[2] = 30
numbers[3] = 40
numbers[4] = 50

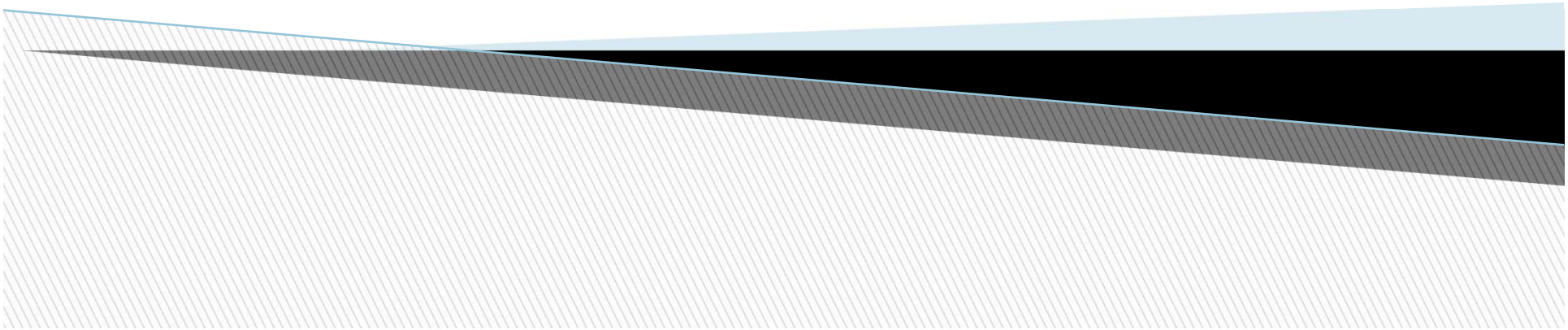


Program using 2-D Array


```
#include <stdio.h>
int main() {
    // Declare and initialize a 2D array
    int arr[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
 // Print the 2D array using a loop
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
return 0;
}
```

Output

1 2 3
4 5 6
7 8 9



String

In C programming, a string is a sequence of characters stored as an array of characters.

Declaration

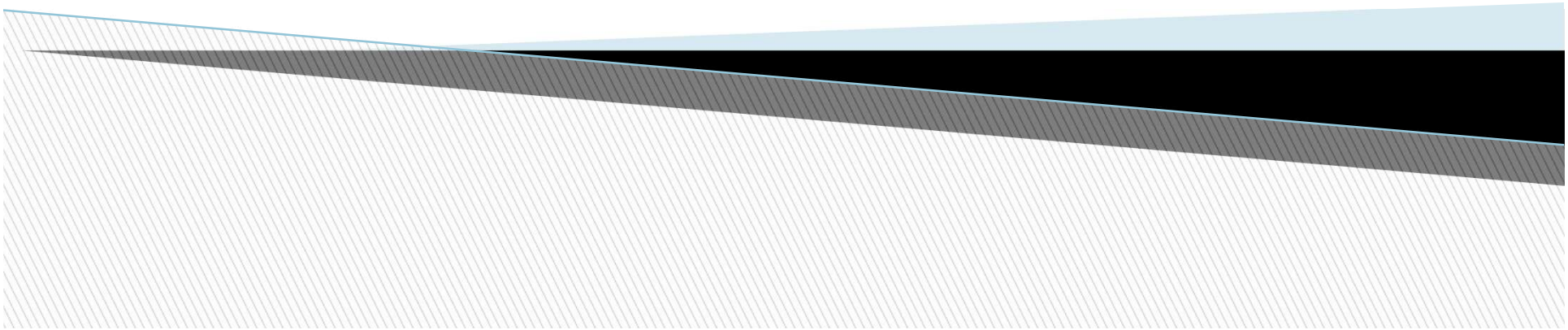
String Declaration: To declare a string in C, you can use the "char" data type followed by array name and square brackets to specify the size of the character array.
Example: char myString[20];

- A null character (\0) is used to denote the end of a string. This is placed at the end of the string.

CONTD..

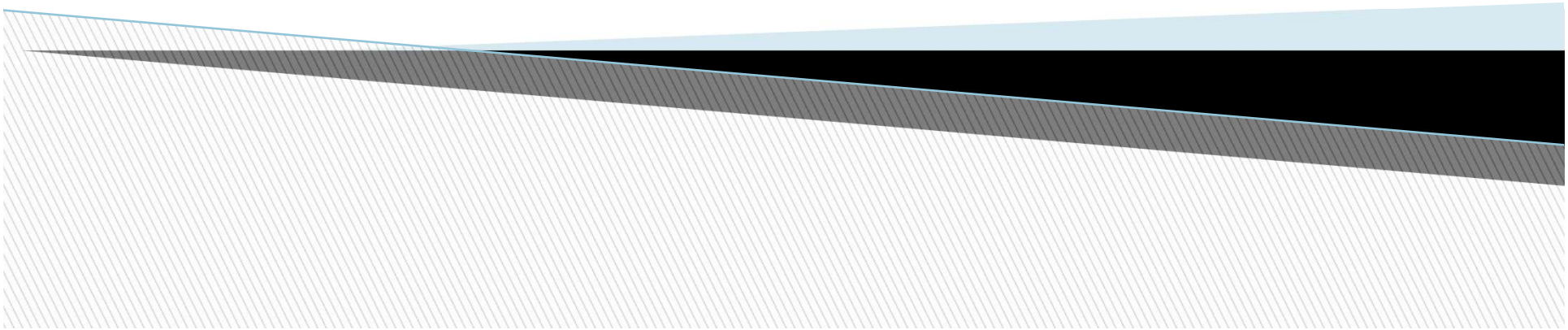
String Initialization: You can initialize a string at the time of declaration.

Example: char greeting[] = "Hello, World!"; **// String initialized at declaration**



CONTD..

Size of a String: The size of a C string (i.e., the number of characters it can hold) is determined by the size of the character array plus one additional byte for the null character. For example, if you have a character array of size 10, it can hold a string of up to 9 characters plus the null character.

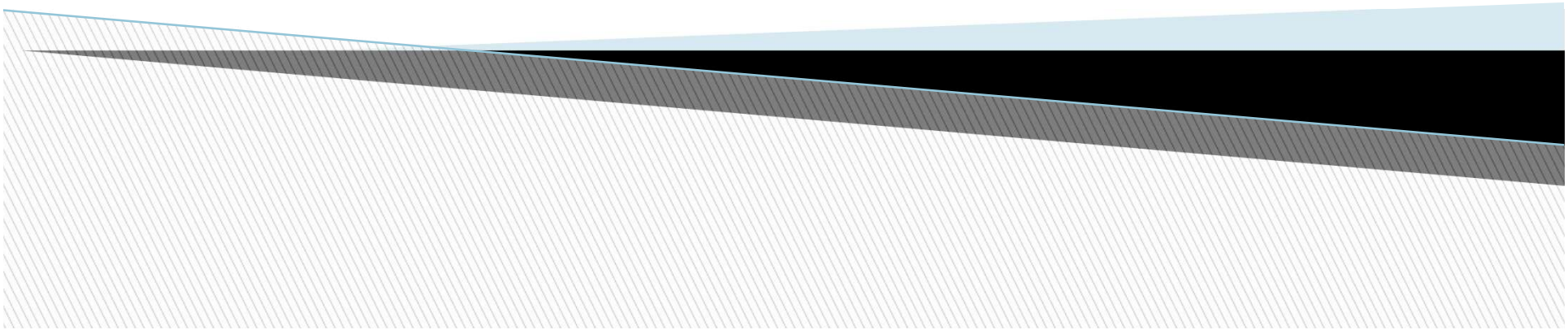


CONTD..

char greeting[] = "Hello, World!";

sizeof(greeting)

Assuming that each character in the string occupies 1 byte (which is typically the case in most systems), the size of the array would be 14 bytes (13 characters + 1 null character).



CONTD..

- ASCII value of null character (\0) is 0
- char arr [] = {'h','e','l','l','o'};
  - In this case, size of arr is 5 because each element is explicitly assigned using single quotation mark. So, This initialization does not include a null terminator.
  - char arr [] = “hello”;
  - In this case, size of arr is 6 because it is stored as a string in memory with a \0 character.

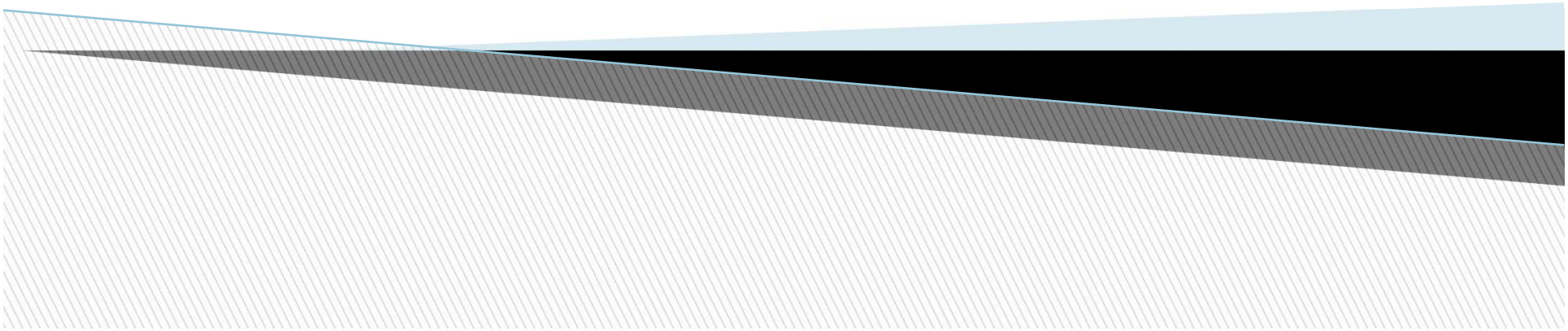
Simple String program

```
#include <stdio.h>
int main() {
int i;
char arr[5] = {'a', 'p', 'p', 'l', 'e'};
for (i = 0; i < 5; i++) {
printf("Value at %d position is %c\n", i, arr[i]);
 }
return 0;
}
```

// here i represents position and arr[i] represents character at position

Output

Value at 0 position is a
Value at 1 position is p
Value at 2 position is p
Value at 3 position is l
Value at 4 position is e



String Functions

- To manipulate strings effectively, Inbuilt functions are there for doing operations on string, such as strlen() - used to find the length of a string.
- These functions can be used by including standard library like <string.h>

Depending on your specific task, you may use these functions.

CONTD..

- **strlen():** Returns the length (number of characters) of a string.

  Syntax: **strlen("Brainware")** returns 9.
- **strcpy():** used to copy the content from source string to target string.

  Syntax: **strcpy(destination, source)**
- **strcat():** This function is used to concatenate (join) two strings.

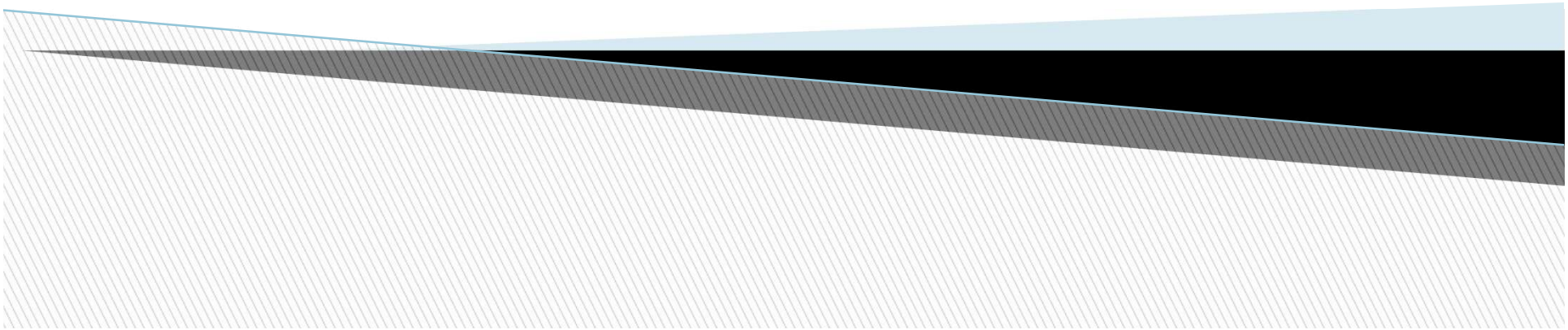
  Syntax: **strcat(string1,string2)** appends the contents of **string2** to the end of **string1.**

String program to find length

```
#include <stdio.h>
#include <string.h>
int main() {
    char str[] = "Hello, World!";
    int length = strlen(str);
    printf("The length of the string is: %d\n",
  length);
  return 0;
}
```

Output..

The length of the string is: 13

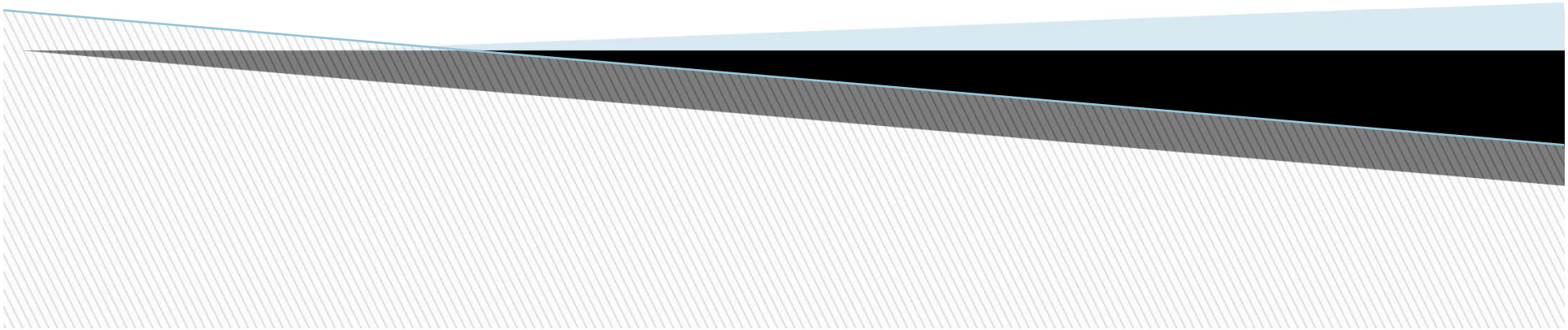


String program to copy the contents of a source string to a destination string:

```
#include <stdio.h>
#include <string.h>
int main() {
    char source[] = "Hello, World!";
    char destination[20]; // Make sure the target string has enough
     space
     strcpy(destination,source);
     printf("Source string: %s\n",source);
     printf("Destination string: %s\n",destination);
     return 0;
}
```

Output..

Source string: Hello, World!
Destination string: Hello, World!



String program to concatenate (join) two strings

```
#include <stdio.h>
#include <string.h>
int main() {
    char str1[50] = "Hello, ";
    char str2[] = "World!";
      strcat(str1, str2);
      printf("Concatenated string: %s\n", str1);
return 0;
}
```

Output..

Concatenated string: Hello, World!