Introduction to Computing and Programming

Strings

Recap

- **Searching**
  - Linear Search
  - Binary Search
- **Sorting**
  - Bubble sort
  - Insertion Sort
  - Selection Sort

Content

- What is String
- Char Array Vs String literals
- String Traversal
- String Pointers
- String Function

Strings

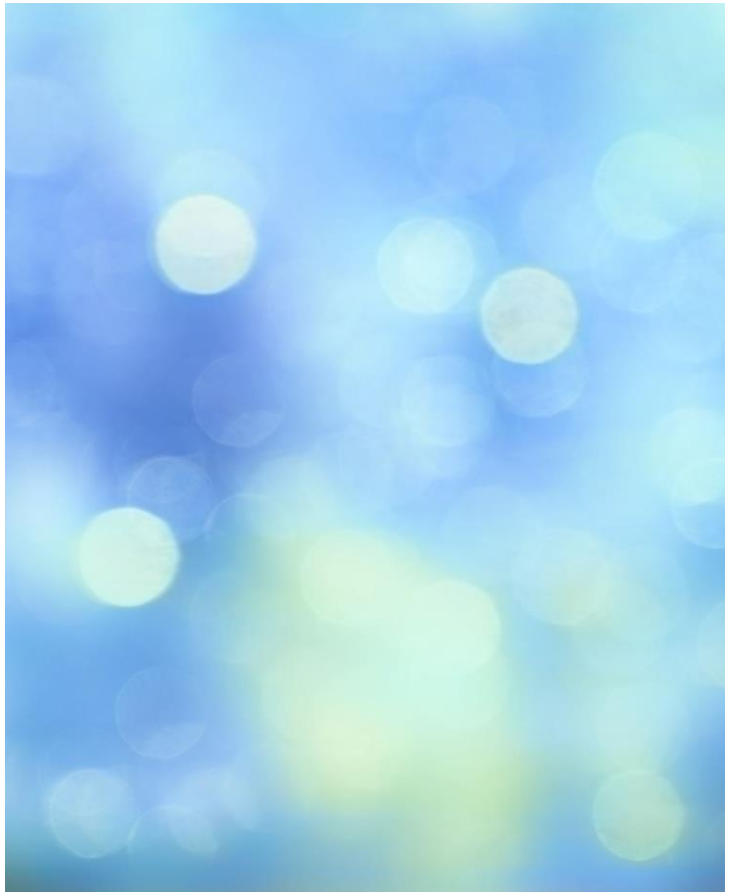
The string can be defined as the **one-dimensional array** of characters terminated by a null ('\0').

The character array or the string is used to manipulate text such as word or sentences.

Each character in the array occupies **one byte** of memory, and the last character must always be **0**.

The **termination character ('\0')** is important in a string since it is the only way to identify where the **string ends**.

When we define a string as char s[10], the character s[10] is **implicitly** initialized with the null in the memory.



s[10] = { }

Using pointers to char (char *str = "Hello";)

String Declaration

- There are **two ways** to declare a string in c language.
  - By **char array**

    char ch[10]={‘s', ‘h', ‘i', ‘v', ‘N', ‘a', ‘d', ‘a', ‘r', '\0'};
  - By **string literal or** **Using pointers to char (char *str = “shivNadar";)**

    char ch[]=“shivNadar”;

```
#include<stdio.h>
#include <string.h>
int main(){
  char ch[10]={'s', 'h', 'i', 'v', 'N', 'a', 'd', 'a', 'r', '\0'};
  char ch2[10]="shivNadar";
  printf("Char Array Value is: %s\n ", ch);
  printf("String Literal Value is: %s\n", ch2);
 return 0;
}
```

Output:
Char Array Value is: shivNadar
String Literal Value is: shivNadar

Difference between char array and string literal

- We need to add the null character '\0' at the end of the array by ourself whereas, it is appended internally by the compiler in the case of the ~~character array~~ string literal.
- The string literal cannot be reassigned to another set of characters whereas, we can reassign the characters of the array.

SL = "hello"

SA = 'hello'

String Literals vs. Character Arrays

1. String Literal: char *str = "Hello";

-Stored in read-only memory (cannot modify individual characters).

-Points to a constant string in memory.

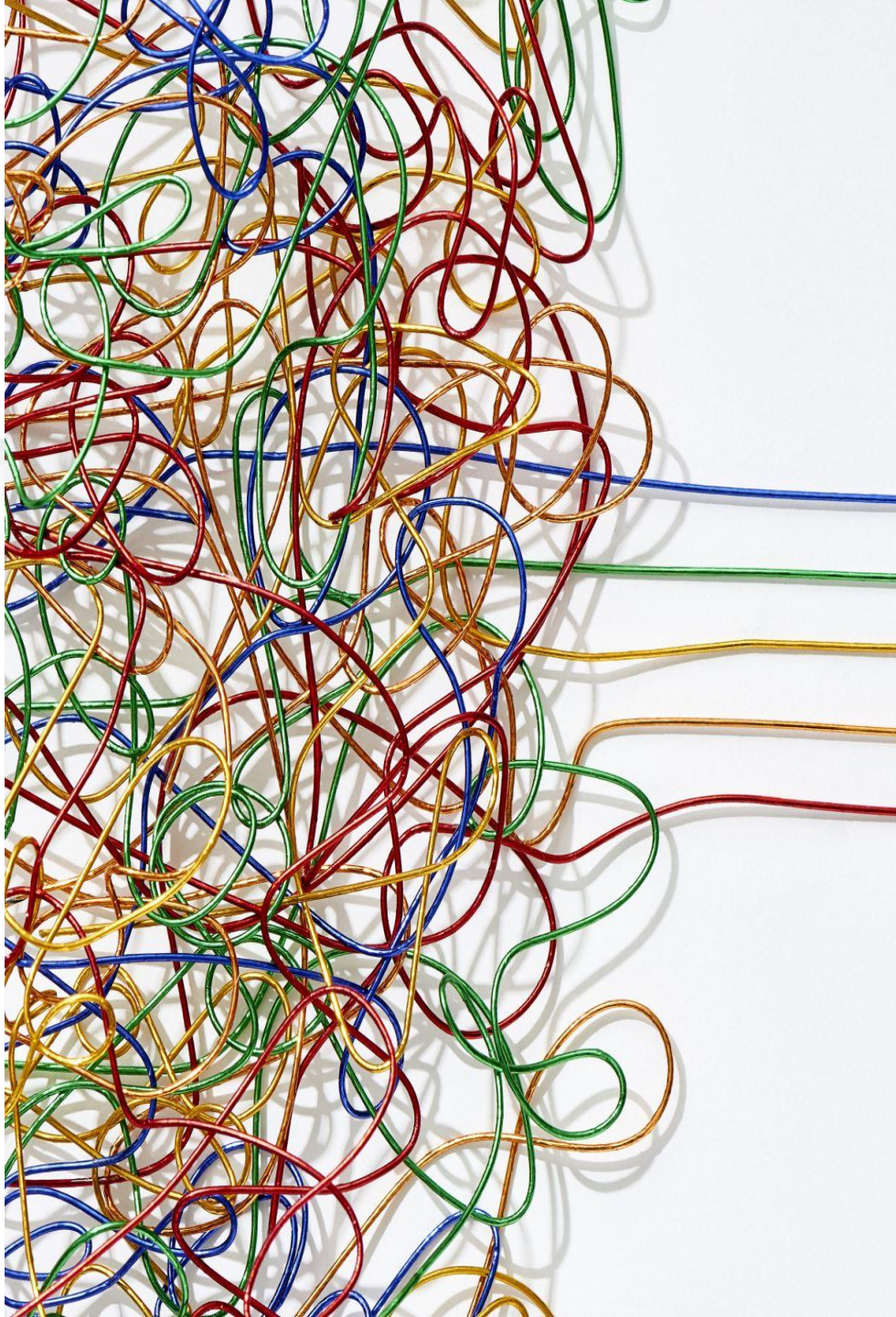
2. Character Array: char str[] = "Hello";

-Stored in modifiable memory (stack).

-Can modify individual characters but cannot reassign the whole array.

String Traversal

- **Traversing** the string is one of the most important aspects.
- We can manipulate a very large text which can be done by traversing the text.
- There are **two ways** to traverse a string.
  - By using the **length of string**
  - By using the **null character**.



```
#include<stdio.h>
void main ()
{  char s[10] = "ShivNadar";
   int i = 0;
   int count = 0;
   while(i<11)
   { if(s[i]=='a' || s[i] == 'e' || s[i] == 'i' || s[i] == 'u' || s[i] == 'o')
   { count ++;  }
      i++; }
   printf("The number of vowels %d",count);
}
```

Output:
The number of vowels 3

```
#include<stdio.h>
void main ()
{  char s[10] = "ShivNadar";
   int i = 0;
   int count = 0;
   while(s[i] != '\0')
   {
      if(s[i]=='a' || s[i] == 'e' || s[i] == 'i' || s[i] == 'u' || s[i] == 'o')
      {  count ++;  }
      i++;  }
   printf("The number of vowels %d",count);
}
```

Output:
The number of vowels 3

Accepting String as Input

```
#include <stdio.h>
int main() {
    char str[100];
      scanf("%[^\n]s",str); // It reads characters until it encounters a newline (\n).
      printf("%s",str);
    return 0;
}
```

Output:
Hello World
Output: Hello World

String Pointer Syntax

- Declaration: char *str;
- Example: char *str = "Hello";
- Memory Layout: str points to the first character of the literal, followed by '\0’.
- Usage: Allows string manipulation using pointer arithmetic.

Advantages of **String Pointers**

- **Efficient Memory Usage**: Only a single pointer rather than copying an entire array.

- **Flexible Reassignment**: char *str = "Hello"; can be reassigned to str = "World";.

- **Pointer Arithmetic**: Easy to navigate through string characters using pointer arithmetic

Modifying Strings Using Pointers

- String Literals: char *str = "Hello";

  Trying str[0] = 'h'; will cause a runtime error (read-only).

gives error without *

- Character Arrays: char str[] = "Hello";
  - **Can modify individual characters like str[0] = 'h'; (works).**

```
char *strLiteral = "Hello";     // Points to a read-only memory area
strLiteral = "World";           // Reassignment is allowed
// strLiteral[0] = 'W';         // This will cause an error

char strArray[] = "Hello";      // Character array in modifiable memory
strArray[0] = 'h';              // Modification is allowed
```

Pointer Arithmetic with Strings

- Concept: Moving through characters with pointers.
- Example Code:

```
char *str = "Hello";
while (*str != '\0') {
    printf("%c ", *str);  // Print each character
    str++;                // Move to the next character
}
```

Explanation: Pointer str moves through each character until '\0'.

Common Pitfalls with String Pointers

- **Modifying String Literals**: Causes runtime errors.
- **Buffer Overflows:** Ensure enough memory is allocated.
- **Null-Termination:** Always remember '\0' in strings, especially with manual manipulations.

Library function: string.h

Function	Function Description
strlen()	Returns the length of the string.
strcpy()	Copy one string to another.
strncpy()	Copy first n characters of one string to another.
strcat()	Concatenates two strings.
strncat()	Concatenates first n characters of one string to another.
strcmp()	Compares two strings.
strncmp()	Compares first n characters of two strings.
strchr()	Find the first occurrence of the given character in the string.
strrchr()	Finds the last occurrence of the given characters in the string.
strstr()	Find the given substring in the string.
strpbrk()	Finds the first occurrence of any of the characters of the given string in the source string.
strtok()	Split the given string into tokens based on some character as a delimiter.

strlen() function

- **strlen()** is used to get the length of a string.
- **sizeof** is used to get the size of a string/array.
- strlen behaves differently, as sizeof also includes the ‘\0’ present in the given string.
- Syntax:
  - strlen(char *str);

```
#include <stdio.h>
#include <string.h>
int main()
{ char alphabet[] =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ";
printf("%d\n", strlen(alphabet));
printf("%d\n", sizeof(alphabet));
char alphabet1[50] =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ";
printf("%d\n", strlen(alphabet1));
printf("%d", sizeof(alphabet1));
return 0;
}
```

Output:

```
26
27
26
50
```

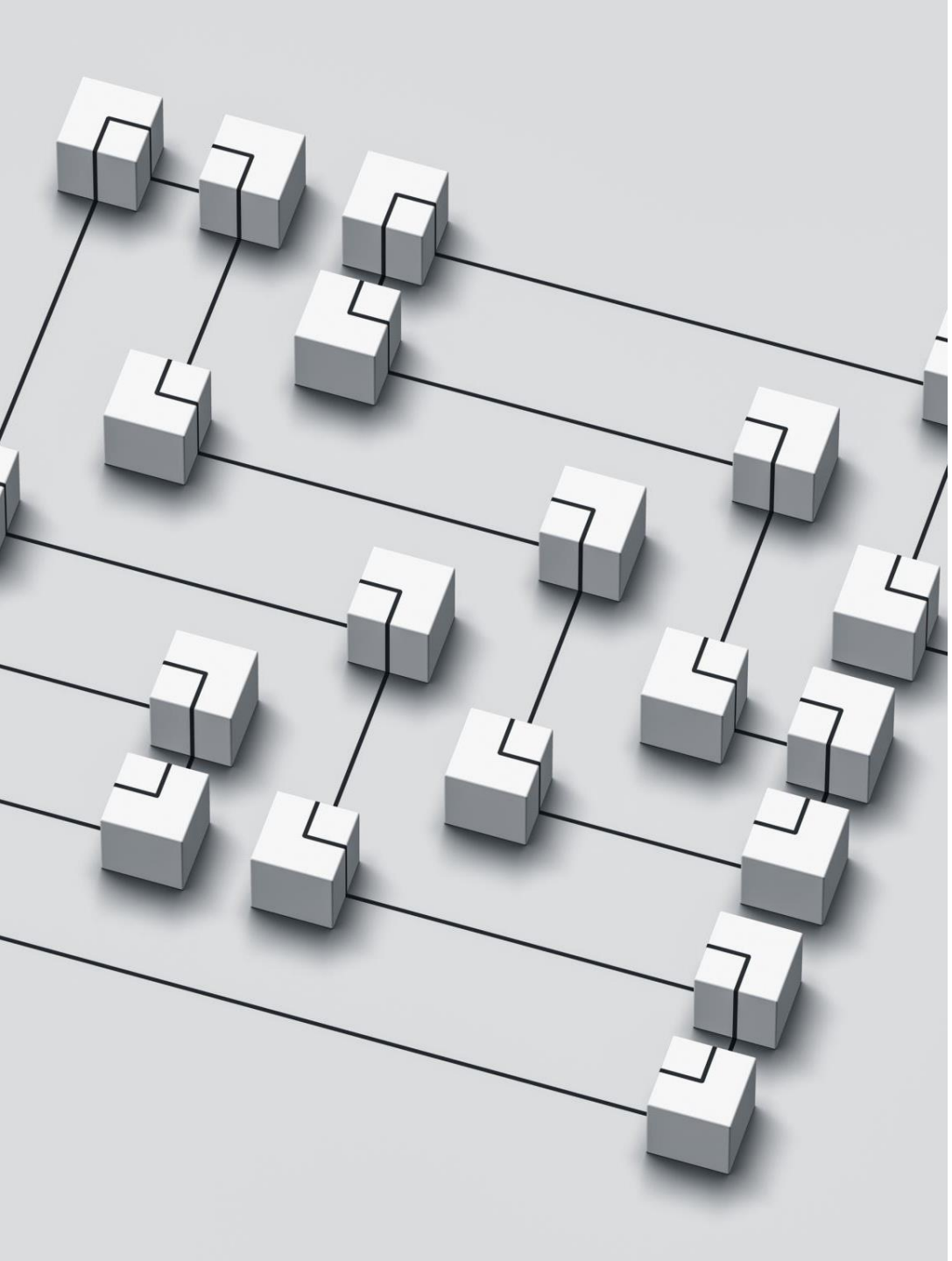
strchr() function

- Used to find the **first occurrence** of a character in a string.
- Checks whether the given character is present in the given string or not.
- If the character is found it returns the pointer to it otherwise it returns a null pointer.
- Syntax:
  - char *strchr(char *str, int ch);

```
#include <stdio.h>
#include <string.h>
int main()
{char* str = "ShivNadar";
    char ch = 'a';
    char* result = strchr(str, ch);
    if (result != NULL) {
        printf("Character '%c' found at position:
%ld\n", ch, result - str);
    }
    else {
        printf("Character '%c' not found.\n", ch);
    }
    return 0;}
```

Output:

Character 'a' found at position: 5



Upcoming Slides

- **String Functions**
- **Structures, Unions and Bit Manipulation**