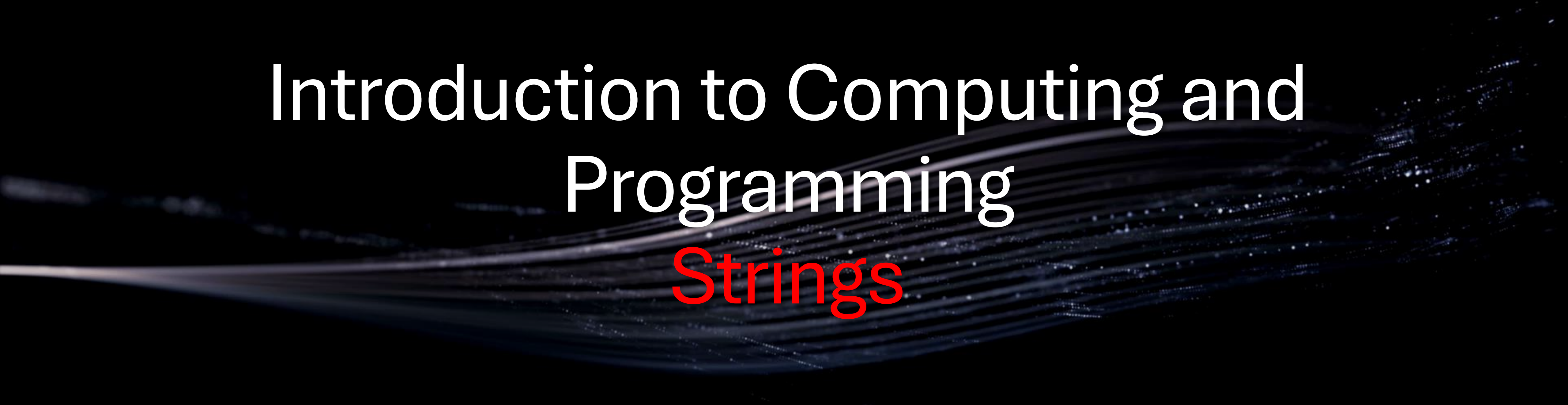
Introduction to Computing and Programming

Strings



Content

- Recap
- String Function
- Array of Pointers to String

Recap

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

Strings

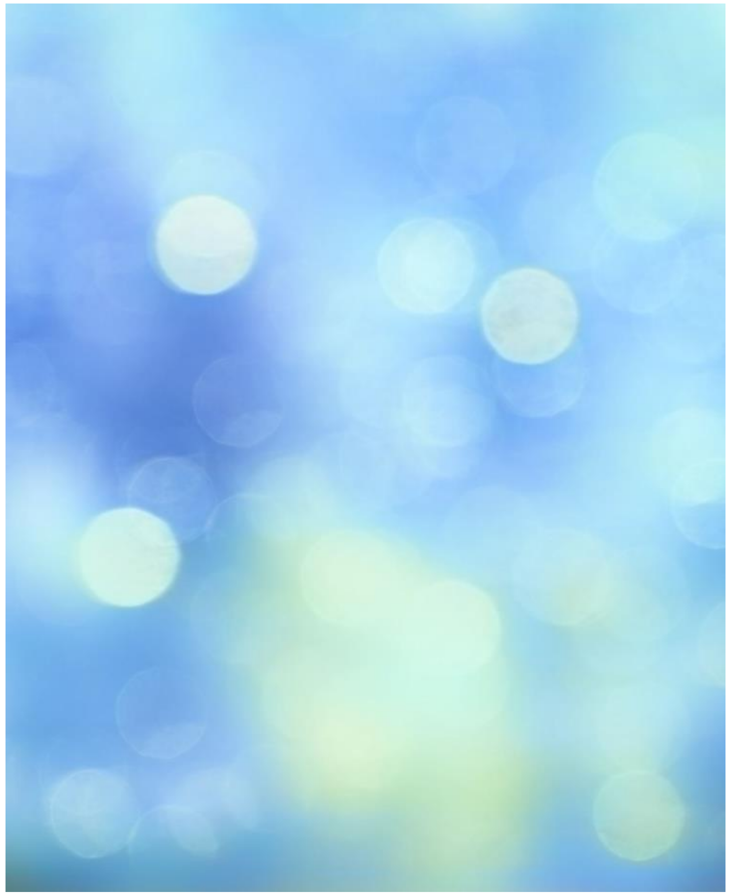
The string can be defined as the **one-dimensional array** of characters terminated by a null ('\0’).

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.



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.

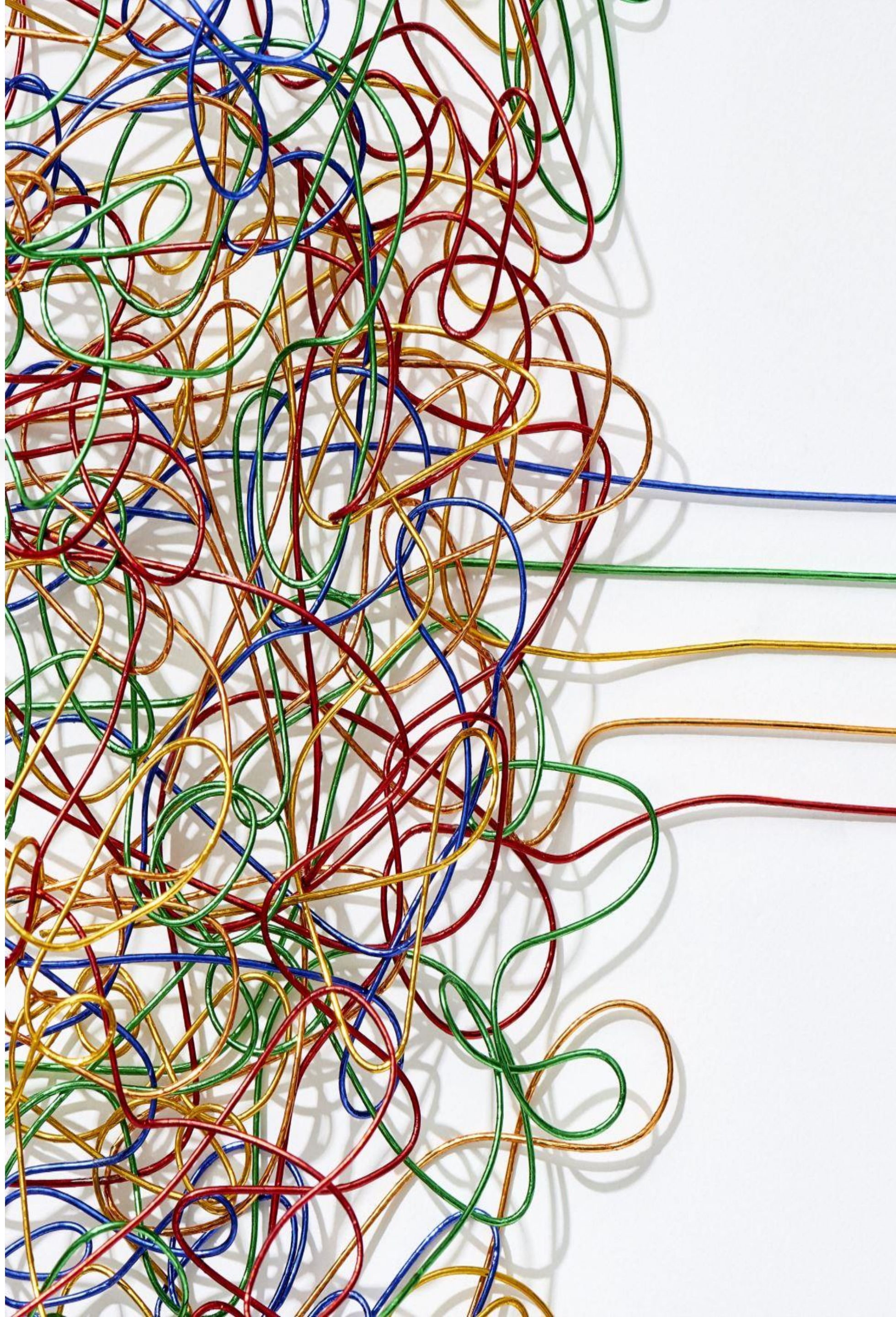
- The string literal cannot be reassigned to another set of characters whereas, we can reassign the characters of the array.

SL = "$\overset{a}{\cancel{h}}$ello"

SA = "hello"

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**.



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.

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
```

strrchr() function

- The **strrchr() function locates the last occurrence of a character** in a string and returns a pointer to it.

- Syntax:
  - char *strrchr(char *str, int ch);

```
#include <stdio.h>
#include <string.h>
int main()
{char* str = "ShivNadar";
   char ch = 'a';
   char* result = strrchr(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: 7
```

strcpy() function

- strcpy **copies a string** from one location to another.
- It takes two arguments: a destination buffer where the copied string will be stored, and a source string that will be copied.
- It copies the entire source string, including the null terminator, into the destination.
- Syntax:
  - char* strcpy(char* destination, char* source);

```
#include <stdio.h>
#include <string.h>
int main()
{
char str1[] = "Hello World!";
char str2[] = "SNU";
char str3[40];
char str4[40];
char str5[] = "SNIoE";
strcpy(str2, str1);
strcpy(str3, "Copy successful");
strcpy(str4, str5);
printf("str1: %s\nstr2: %s\nstr3:
%s\nstr4:%s\n", str1, str2, str3, str4);
return 0;
}
```

Output:

```
str1: o World!
str2: Hello World!
str3: Copy successful
str4:SNIoE
```

strncpy() function

- It copies the first n characters from one string into the memory of another string.
- This does not add a null terminating character to the copied data, so make sure that the destination string has a null terminating character somewhere after the copied data.
- Syntax:
  - strncpy(char * destination, char * source, size_t n);

```
#include <stdio.h>
#include <string.h>
int main()
{
char str1[] = "Hello World!";
char str2[] = "~~Write~~ code!";
strncpy(str2, str1, 6);
printf("%s\n", str1);
printf("%s\n", str2);

            return 0;
}
```

Output:

```
Hello World!
Hello code!
```

strcat() function

- It appends the string pointed to by src to the end of the string pointed to by dest.
- It will append a copy of the source string in the destination string. plus a terminating Null character.
- The initial character of the string(src) overwrites the Null-character present at the end of the string(dest).

- Syntax:
  - char *strcat(char *dest, char *src);

```
#include <stdio.h>
#include <string.h>
int main()
{
char example[100];
strcat(example, "Hello ");
strcat(example, "World!!!");
printf("%s\n", example);
return 0;
}
```

Output:

```
Hello World!!!
```

strncat() function

- It appends n characters from the src string to the end of desc string plus a terminating Null-character.
- The initial character of the string(src) overwrites the Null-character present at the end of a string(dest).
- The length of the string(dest) becomes strlen(dest)+n.
- Syntax:
  - char *strncat(char *dest, const char *src, size_t n)

```
#include <stdio.h>
#include <string.h>
int main()
{
char str1[] = "Hello World!";
char str2[] = "Write code!";
strncat(str1, str2, 6);
printf("%s\n", str1);
printf("%s\n", str2);

        return 0;
}
```

Output:

```
Hello World!Write
Write code!
```

strcmp() function

- C strcmp() function works by comparing the two strings lexicographically.
- It means that it compares the ASCII value of each character till the non-matching value is found or the NULL character is found.

```
#include<stdio.h>
#include<string.h>
int main()
{
    char first_str[] = "SNU";
    char second_str[] = "SNU";
    int res = strcmp(first_str, second_str);
    if (res==0)
        printf("Strings are equal");
    else
        printf("Strings are unequal");
    printf("\nValue returned by strcmp()
is: %d" , res);
    return 0;
}
```

Output:

```
Strings are equal
Value returned by strcmp() is: 0
```

strcmp() function Cont…

If a non-matching character is found,

- If the ASCII value of the character of the first string is greater than that of the second string, then the positive difference (>0) between their ASCII values is returned.
- If the ASCII value of the character of the first string is less than that of the second string, then the negative difference (< 0) between their ASCII values is returned.

```
#include<stdio.h>
#include<string.h>
int main()
{  char first_str[] = "SNU";
   char second_str[] = "ZNU";
   char third_str[] = "CNU";
   int res = strcmp(first_str, second_str);
   int res1 = strcmp(first_str, third_str);
   if (res==0 && res1==0)
      printf("Strings are equal");
   else
      printf("Strings are unequal");
   printf("\nValue returned by strcmp() is: %d" , res);
   printf("\nValue returned by strcmp() is: %d" , res1);
   return 0;
}
```

Output:

```
Strings are unequal
Value returned by strcmp() is: -7
Value returned by strcmp() is: 16
```

S - Z

strncmp() function

- If n comparisons have been made without any mismatches then the function returns zero.
- If the end of both strings has been reached without any mismatches then the function returns zero.
- If the ASCII value of the character of the first string is greater than that of the second string, then the positive difference (>0) between their ASCII values is returned.
- If the ASCII value of the character of the first string is less than that of the second string, then the negative difference (< 0) between their ASCII values is returned.

```
#include<stdio.h>
#include<string.h>
int main()
{
    char first_str[] = "SNU";
    char second_str[] = "SNIoE";
    char third_str[] = "SNUniv";
    int res = strncmp(first_str, second_str,4);
    int res1 = strncmp(first_str, third_str,3);
    printf("\nValue returned by strcmp() is: %d" , res);
    printf("\nValue returned by strcmp() is: %d" , res1);
    return 0;
}
```

[illegible]

Strcmpi( )

Output:

```
Value returned by strcmp() is: 12
Value returned by strcmp() is: 0
```

SNU Snu

strstr() function

- This function takes two strings s1 and s2 as arguments and finds the first occurrence of the string s2 in the string s1.
- s1: This is the main string to be examined.
- s2: This is the sub-string to be searched in string.
- Syntax:
  - char *strstr (char *s1, char *s2);

```
#include <stdio.h>
#include <string.h>
int main()
{  char s1[] = "SNUniv";
   char s2[] = "SNU";
   char s3[] = "SNZ";
   char* p;
   char* p1;
   p = strstr(s1, s2);
   p1 = strstr(s1, s3);
   if (p || p1) {
      printf("string '%s' in '%s' is ““'%s'\n",s2, s1, p);
      printf("string '%s' in '%s' is ""'%s'\n",s3, s1, p1);
   }
}
```

Output:

```
string ‘SNU' in 'SNUniv' is 'SNUniv'
string 'SNZ' in 'SNUniv' is '(null)'
```

strpbrk() function

- This function finds the first character in the string s1 that matches any character specified in s2 (It excludes terminating null-characters).
- s1 : string to be scanned.
- s2 : string containing the characters to match.
- Syntax:
  - char *strpbrk(char *s1, char *s2)

```
#include <stdio.h>
#include <string.h>
int main()
{  char s1[] = "SNUniv";
   char s2[] = "SNU";
   char s3[] = "NCR";
   char* p;
   char* p1;
   p = strpbrk(s1, s2);
   p1 = strpbrk(s1, s3);
   if (p!=0 || p1!=0) {
      printf("First matching char %c\n",*p);
      printf("First matching char %c",*p1);
   }
   return 0;
}
```

Output:

```
First matching char S
First matching char N
```

strtok() function

- It splits str[] according to given delimiters and returns the next token.
- It needs to be called in a loop to get all tokens. It returns NULL when there are no more tokens.
- str: It is the pointer to the string to be tokenized.
- delims: It is a string containing all delimiters.
- Syntax:
  - char *strtok(char *str, char *delims);

Example 1

```
#include <stdio.h>
#include <string.h>
int main()
{
char str[] = "CSD101,CSD102,CSD103";
char* token = strtok(str, ",");
while (token != NULL) {
printf(" % s\n", token);
token = strtok(NULL, ",");
}
return 0;
}
```

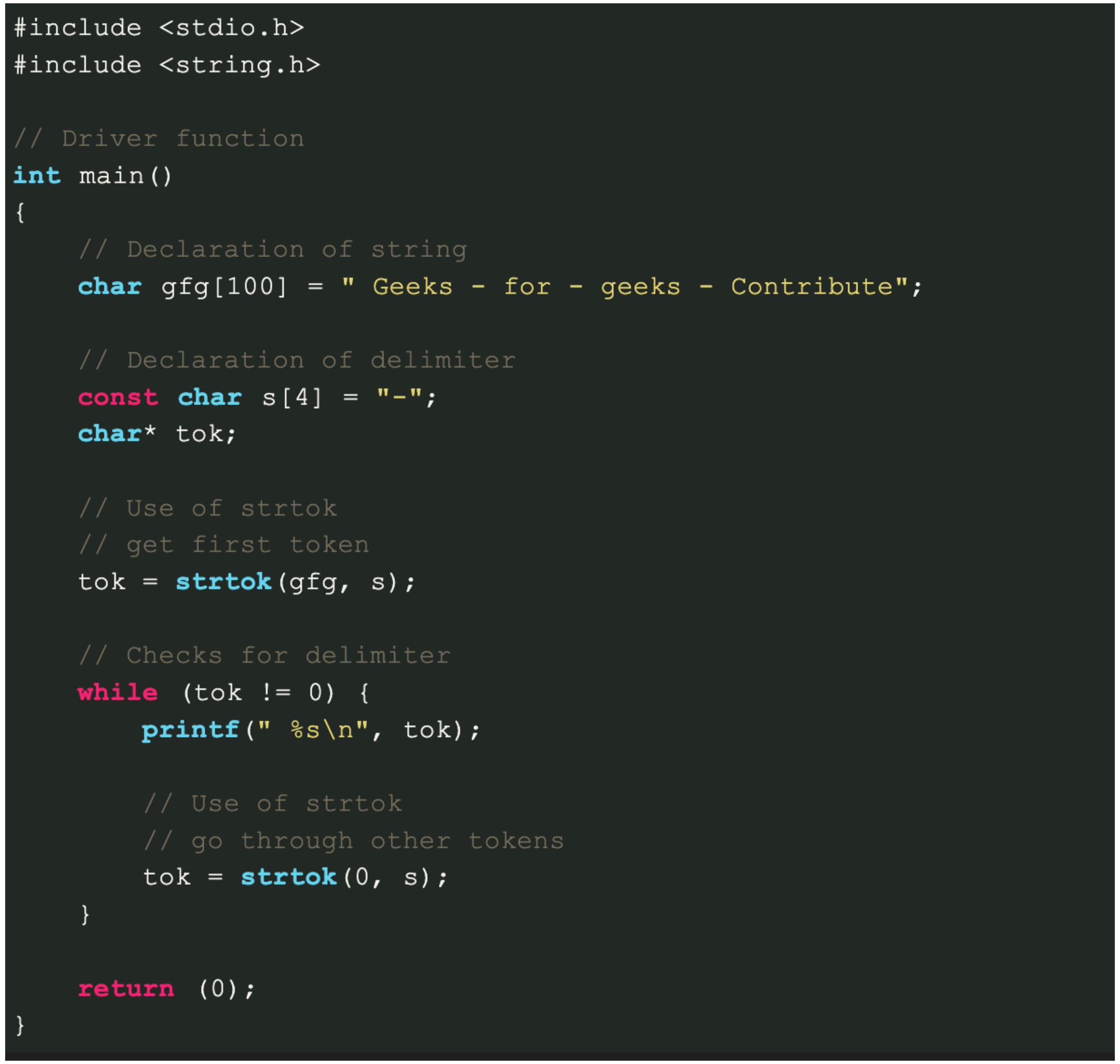
Output:

```
CSD101
CSD102
CSD103
```

RETURNS

1. It returns the pointer to the first token encountered in the string.
2. It returns NULL if there are no more tokens found.

Example 2- Program to demonstrates the use of the strtok() function to tokenize a string based on a delimiter.



```c
#include <stdio.h>
#include <string.h>

// Driver function
int main()
{
    // Declaration of string
    char gfg[100] = " Geeks - for - geeks - Contribute";

    // Declaration of delimiter
    const char s[4] = "-";
    char* tok;

    // Use of strtok
    // get first token
    tok = strtok(gfg, s);

    // Checks for delimiter
    while (tok != 0) {
        printf(" %s\n", tok);

        // Use of strtok
        // go through other tokens
        tok = strtok(0, s);
    }

    return (0);
}
```

strerror() function

- The strerror() function returns a string describing the meaning of error code.
- Syntax:
  - strerror(int errorcode)
- errorcode required an error code.
- A pointer to a string describing the meaning of the error code.

```
#include <stdio.h>
#include <string.h>

int main() {
  printf("%s\n", strerror(0));
  printf("%s\n", strerror(1));
  printf("%s\n", strerror(2));
  printf("%s\n", strerror(3));
  return 0;
}
```

Output:

```
Success
Operation not permitted
No such file or directory
No such process
```

Strings - Special Characters

- Strings must be written within quotes, C will misunderstand this string, and generate an error:
  - char txt[] = "We are the so-called "Vikings" from the north.";
- The solution to avoid this problem, is to use the backslash escape character.
  - The **backslash (\) escape character turns** special characters into string characters:

Escape character	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

Examples

The sequence \" inserts a double quote in a string:

Example

```
char txt[] = "We are the so-called \"Vikings\" from the north.";
```

The sequence \' inserts a single quote in a string:

Example

```
char txt[] = "It\'s alright.";
```

The sequence \\ inserts a single backslash in a string:

Example

```
char txt[] = "The character \\ is called backslash.";
```

Other Escape characters are:

Escape Character	Result
\n	New Line
\t	Tab
\0	Null

Read string from the user using scanf

- We can use the scanf() function to read a string.
- The scanf() function reads the sequence of characters until it encounters whitespace (space, newline, tab, etc.).
- At the time of scanning the name, name is taken instead of &name in scanf function.
    - This is because name is a char array.
- The name in scanf() **already points to the address of the first element** in the string, which is why we don't need to use &.

```
#include <stdio.h>
int main()
{
    char name[20];
    printf("Enter name: ");
    scanf("%s", name);
    printf("Your name is %s.", name);
    return 0;
}
```

```
Enter name: Dennis Ritchie
Your name is Dennis.
```

Read string with spaces using gets

- gets() has been removed from c11. So, there might be some warnings when implementing it.
- We see here that this command doesn't care about the size of the array. So, there is a possibility of **Buffer Overflow.**

```
#include <stdio.h>
int main()
{
  char str[20];
  gets(str);
  printf("%s", str);
  return 0;
}
```

```
Hello World
Hello World
```

Read string with spaces using scanf

- Syntax: scanf(“%[^\n]%*c”, str);
- Here, [] is the scanset character. ^\n tells to take input until newline is not found.
- Then, with this %*c , it reads the newline character and here * is used to indicate that this newline character is discarded.

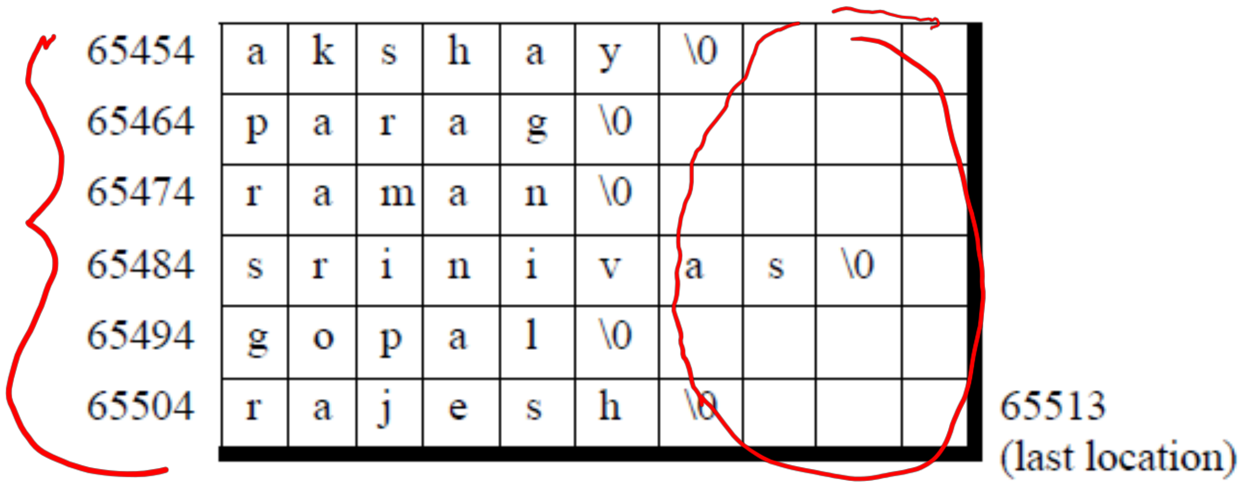
```
#include <stdio.h>
int main()
{
   char str[20];
   scanf("%[^\n]%*c", str);
   printf("%s", str);

   return 0;
}
```

```
Hello World
Hello World
```

Two-Dimensional Array of Characters

- The order of the subscripts in the array declaration is important.
- The first subscript gives the number of names in the array, while the second subscript gives the length of each item in the array.
- Example:

```
{
char masterlist[6][10] = {
"akshay",
"parag",
"raman",
"srinivas",
"gopal",
"rajesh"
} ;
```



65454	a	k	s	h	a	y	\0			
65464	p	a	r	a	g	\0				
65474	r	a	m	a	n	\0				
65484	s	r	i	n	i	v	a	s	\0	
65494	g	o	p	a	l	\0				
65504	r	a	j	e	s	h	\0			

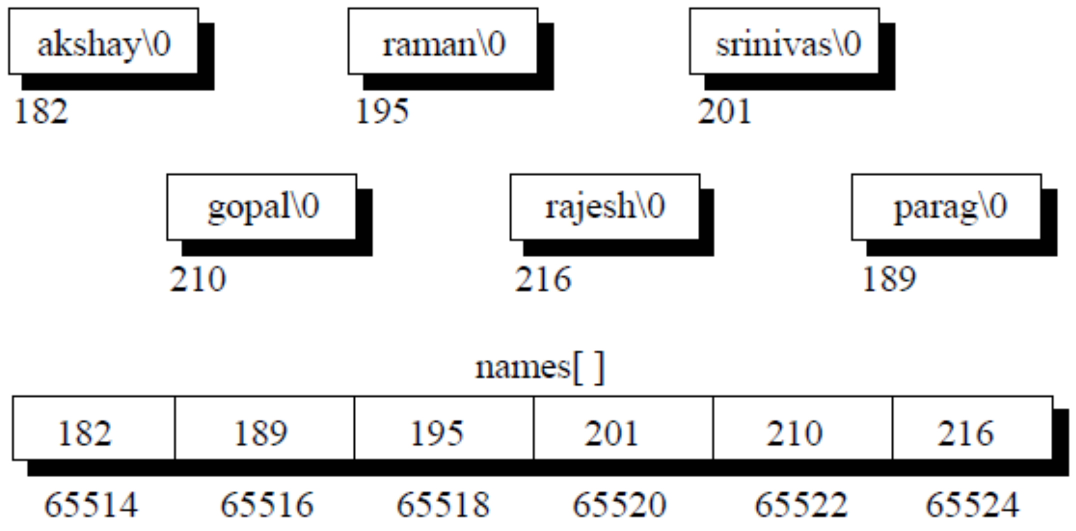
65513 (last location)

Array of Pointers to Strings

- A pointer variable always contains an address.
- Therefore, if we construct an array of pointers it would contain a number of addresses.

```
char *names[ ] = {
"akshay",
"parag",
"raman",
"srinivas",
"gopal",
"rajesh"
} ;
```

- In this declaration **names[ ]** is an array of pointers. It contains base addresses of respective names. That is, base address of "akshay" is stored in **names[0]**, base address of "parag" is stored in **names[1]** and so on.



Benefits of Array of pointers over array of characters

- An array of pointers make a more efficient use of available memory.
  - Example: In the two-dimensional array of characters, the strings occupied 60 bytes. As against this, in array of pointers, the strings occupy only 41 bytes—a net saving of 19 bytes.
- An array of pointers to store strings is to obtain greater ease in manipulation of the strings.
  - Example is shown in the upcoming slide

Exchange names using 2-D array of characters

```
#include<stdio.h>
#include<string.h>
void main( )
{char names[ ][10] = {
"akshay",
"parag",
"raman",
"srinivas",
"gopal",
"rajesh"} ;
int i ;
char t ;
printf ( "\nOriginal: %s %s", &names[2][0],
&names[3][0] ) ;
```

```
for ( i = 0 ; i <= 9 ; i++ )
{
t = names[2][i] ;
names[2][i] = names[3][i] ;
names[3][i] = t ;
}
printf ( "\nNew: %s %s", &names[2][0],
&names[3][0] ) ;}
```

Output:

```
Original: raman srinivas
New: srinivas raman
```

To exchange the names, we are required to exchange corresponding characters of the two names. In effect, 10 exchanges are needed to interchange two names.

Exchange names using array of pointers to string

```
#include<stdio.h>
#include<string.h>
void main( )
{
char *names[ ] = {
"akshay",
"parag",
"raman",
"srinivas",
"gopal",
"rajesh"
} ;
char *temp ;
printf ( "Original: %s %s", names[2], names[3] ) ;
temp = names[2] ;
names[2] = names[3] ;
names[3] = temp ;
printf ( "\nNew: %s %s", names[2], names[3] ) ;
}
```

Output:

```
Original: raman srinivas
New: srinivas raman
```

we are required to do is exchange the addresses (of the names) stored in the array of pointers, rather than the names themselves. Thus, by effecting just one exchange we are able to interchange names. This makes handling strings very convenient.

Limitation of Array of Pointers to Strings

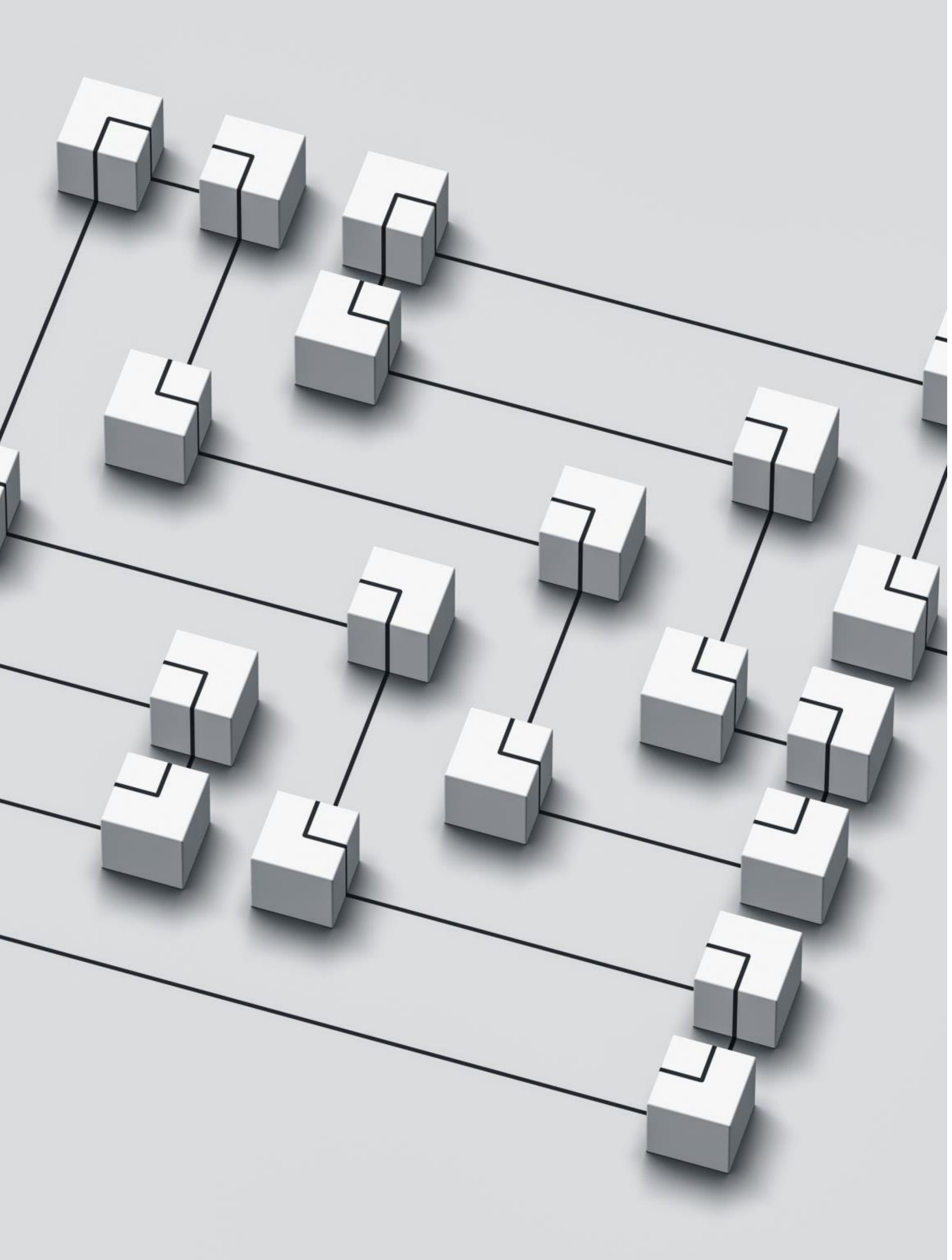
- We can initialize the strings at the place where we are declaring the array, but we cannot receive the strings from keyboard using **scanf( )**.
- The program will not work because; when we are declaring the array it is containing garbage values.
- The garbage values can not be sent as an address for scanf( ) function

```
main( )
{
char *names[6] ;
int i ;
for ( i = 0 ; i <= 5 ; i++ )
{
printf ( "\nEnter name " ) ;
scanf ( "%s", names[i] ) ;
}
}
```

Solution: Use Dynamic Memory allocation

```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
void main()
{
char *names[6] ;
char n[50] ;
int len, i ;
char *p ;
for ( i = 0 ; i <= 5 ; i++ )
{
printf ( "\nEnter name " ) ;
scanf ( "%s", n ) ;
len = strlen ( n ) ;
p = malloc ( len + 1 ) ;
strcpy ( p, n ) ;
names[i] = p ;
}
for ( i = 0 ; i <= 5 ; i++ )
printf ( "\n%s", names[i] ) ;
}
```

```
Enter name Name1
Enter name Name2
Enter name Name3
Enter name Name4
Enter name Name5
Enter name Name6

Name1
Name2
Name3
Name4
Name5
Name6
```



Upcoming Slides

- **Structures, Unions and Bit Manipulation**