Programming and Algorithms

COMP1038.PGA

Week 6 – Lecture 1 & 2: Characters and Strings

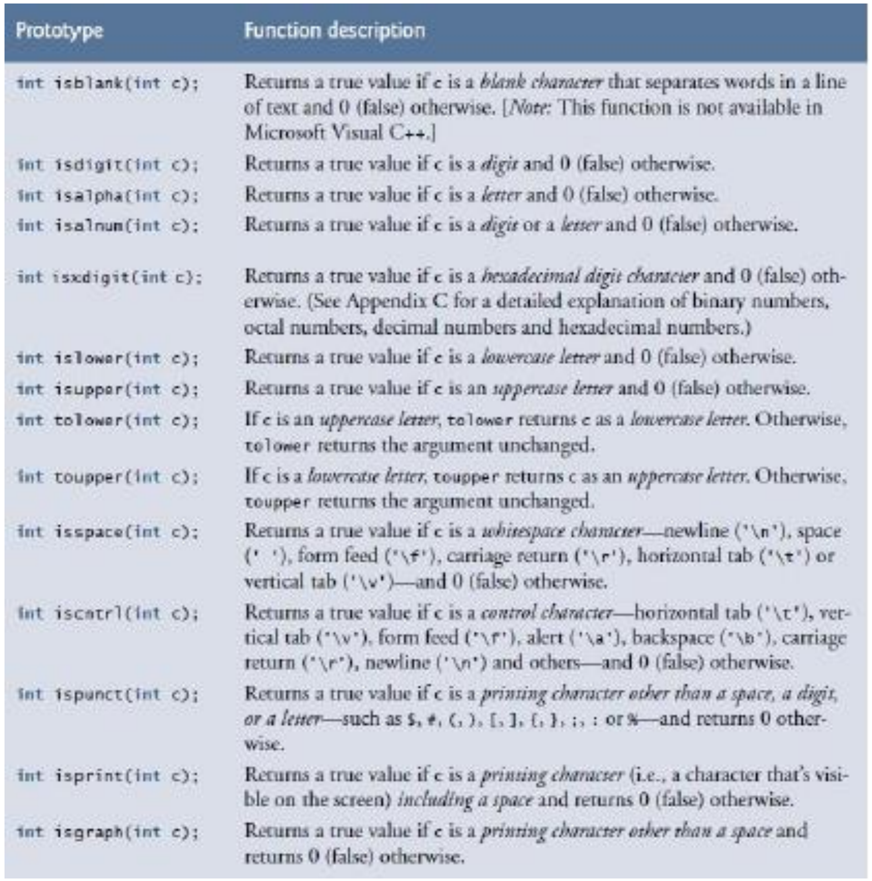
Dr. Pushpendu Kar

Outline

- Character handling
- Introduction to strings
- Declaration of strings
- Initializing of strings
- Reading strings
- Writing strings
- String handling functions
- Array of strings
- Conclusion

Characters handling

- Character handling functions are inside *ctype.h* header file. So you need to include this header file in your program to use these functions.



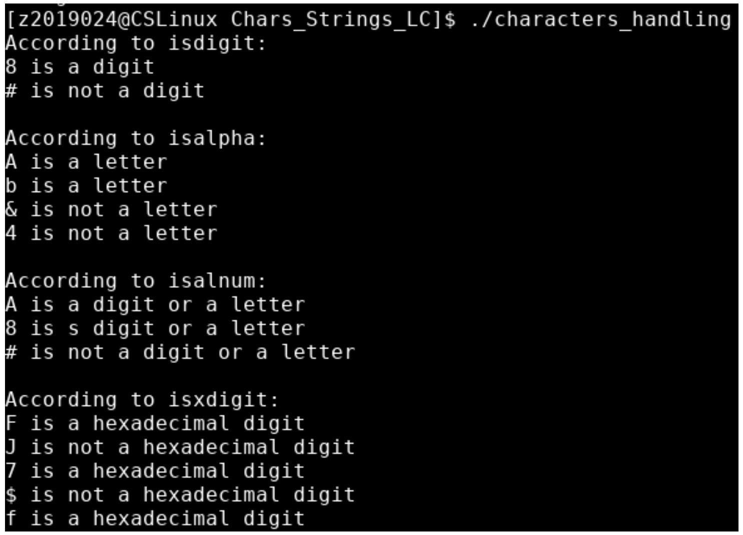
Prototype	Function description
`int isblank(int c);`	Returns a true value if `c` is a *blank character* that separates words in a line of text and 0 (false) otherwise. [*Note:* This function is not available in Microsoft Visual C++.]
`int isdigit(int c);`	Returns a true value if `c` is a *digit* and 0 (false) otherwise.
`int isalpha(int c);`	Returns a true value if `c` is a *letter* and 0 (false) otherwise.
`int isalnum(int c);`	Returns a true value if `c` is a *digit* or a *letter* and 0 (false) otherwise.
`int isxdigit(int c);`	Returns a true value if `c` is a *hexadecimal digit character* and 0 (false) otherwise. (See Appendix C for a detailed explanation of binary numbers, octal numbers, decimal numbers and hexadecimal numbers.)
`int islower(int c);`	Returns a true value if `c` is a *lowercase letter* and 0 (false) otherwise.
`int isupper(int c);`	Returns a true value if `c` is an *uppercase letter* and 0 (false) otherwise.
`int tolower(int c);`	If `c` is an *uppercase letter*, `tolower` returns `c` as a *lowercase letter*. Otherwise, `tolower` returns the argument unchanged.
`int toupper(int c);`	If `c` is a *lowercase letter*, `toupper` returns `c` as an *uppercase letter*. Otherwise, `toupper` returns the argument unchanged.
`int isspace(int c);`	Returns a true value if `c` is a *whitespace character*—newline (`'\n'`), space (`' '`), form feed (`'\f'`), carriage return (`'\r'`), horizontal tab (`'\t'`) or vertical tab (`'\v'`)—and 0 (false) otherwise.
`int iscntrl(int c);`	Returns a true value if `c` is a *control character*—horizontal tab (`'\t'`), vertical tab (`'\v'`), form feed (`'\f'`), alert (`'\a'`), backspace (`'\b'`), carriage return (`'\r'`), newline (`'\n'`) and others—and 0 (false) otherwise.
`int ispunct(int c);`	Returns a true value if `c` is a *printing character other than a space, a digit, or a letter*—such as `$`, `#`, `(`, `)`, `[`, `]`, `{`, `}`, `;`, `:` or `%`—and returns 0 otherwise.
`int isprint(int c);`	Returns a true value if `c` is a *printing character* (i.e., a character that's visible on the screen) *including a space* and returns 0 (false) otherwise.
`int isgraph(int c);`	Returns a true value if `c` is a *printing character other than a space* and returns 0 (false) otherwise.

Source: Deitel and Deiltel(2016). C How to Program with an Introduction to C++ (8th Ed.). Pearson.

The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

Characters handling cont…

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

int main(){
  printf("%s\n%s%s\n%s%s\n\n", "According to isdigit: ",
    isdigit('8') ? "8 is a " : "8 is not a ", "digit",
    isdigit('#') ? "# is a " : "# is not a ", "digit");

  printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n", "According to isalpha: ",
    isalpha('A') ? "A is a " : "A is not a ", "letter",
    isalpha('b') ? "b is a " : "b is not a ", "letter",
    isalpha('&') ? "& is s " : "& is not a ", "letter",
    isalpha('4') ? "4 is a " : "4 is not a ", "letter");

  printf("%s\n%s%s\n%s%s\n%s%s\n\n", "According to isalnum: ",
    isalnum('A') ? "A is a " : "A is not a ", "digit or a letter",
    isalnum('8') ? "8 is s " : "8 is not a ", "digit or a letter",
    isalnum('#') ? "# is a " : "# is not a ", "digit or a letter");

  printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n%s%s\nn", "According to isxdigit: ",
    isxdigit('F') ? "F is a " : "F is not a ", "hexadecimal digit",
    isxdigit('J') ? "J is s " : "J is not a ", "hexadecimal digit",
    isxdigit('7') ? "7 is a " : "7 is not a ", "hexadecimal digit",
    isxdigit('$') ? "$ is a " : "$ is not a ", "hexadecimal digit",
    isxdigit('f') ? "f is a " : "f is not a ", "hexadecimal digit");

  return(0);
}
```



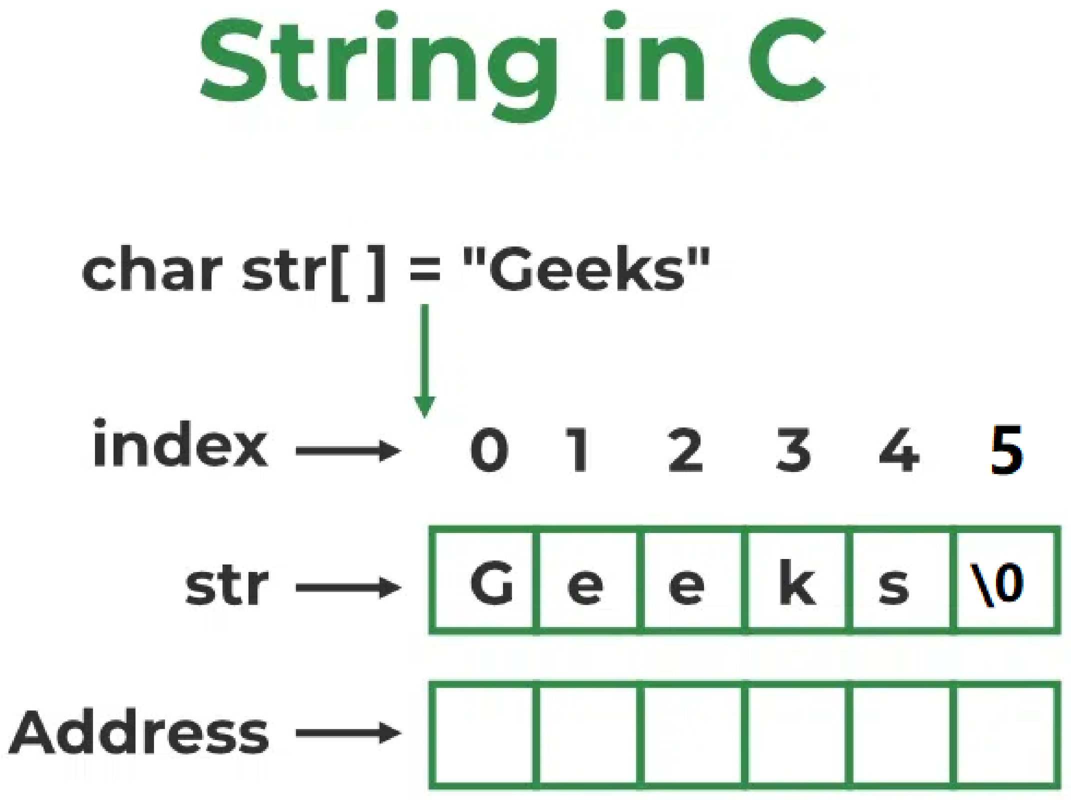
```
[z2019024@CSLinux Chars_Strings_LC]$ ./characters_handling
According to isdigit:
8 is a digit
# is not a digit

According to isalpha:
A is a letter
b is a letter
& is not a letter
4 is not a letter

According to isalnum:
A is a digit or a letter
8 is s digit or a letter
# is not a digit or a letter

According to isxdigit:
F is a hexadecimal digit
J is not a hexadecimal digit
7 is a hexadecimal digit
$ is not a hexadecimal digit
f is a hexadecimal digit
```

Source: Dola saha, C programming for engineer, 2017.

Introduction to strings

- The C language does not have a specific "String" data type, the way some other languages such as C++ and Java do.
- In C language, String is a 1-d array of type char.
- By convention, a string in C is terminated by the end-of-string sentinel '\0' [backslash zero](null character).
- The difference between a character array and a C string is the string is terminated with a unique character '\0'.

Introduction cont...



Source: https://www.geeksforgeeks.org/

String Declaration

- Declaring a string in C is as simple as declaring a one-dimensional array.
  char string_name[size];
- In the above syntax *str_name* is any name given to the string variable and *size* is used to define the length of the string, i.e the number of characters the string will store.
- There is an extra terminating character which is the **Null character ('\0') used to indicate the termination of a string that differs strings from normal character arrays.**

String Literals

- String literal values are represented by sequences of characters between double quotes (")
- Examples
  - "" - empty string
  - "hello" - a string literal
- "a" versus 'a'
  - 'a' is a single character value (stored in 1 byte) as the ASCII value for a
  - "a" is an array with two characters, the first is a, the second is the character value \0
- String literal is an array, can refer to a single character from the literal as a character.
- Example:

  ```
  printf("%c", "hello"[1]);
  ```

  outputs the character 'e'
- During compilation, C creates space for each string literal (# of characters in the literal + 1)
  - referring to the literal refers to that space (as if it is an array)

String Initialization

A string in C can be initialized in different ways. Below are the examples to declare a string with the name *str* and initialize it with “Nottingham”.

1. **Assigning a string literal without size**
   - String literals can be assigned without size. Here, the name of the string *str* acts as a pointer because it is an array.

     char str[ ] = “Nottingham”;
2. **Assigning a string literal with a predefined size**
   - String literals can be assigned with a predefined size. But we should always account for one extra space which will be assigned to the null character. If we want to store a string of size *n* then we should always declare a string with a size equal to or greater than n+1.

     char str[50] = “Nottingham”;
3. **Assigning character by character with size**
   - We can also assign a string character by character. But we should remember to set the end character as ‘\o’ which is a null character.

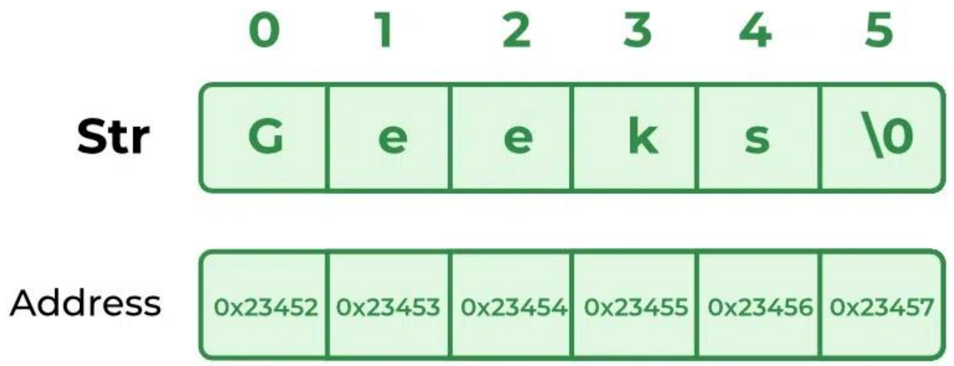
     char str[11] = { 'N','o','t','t','i','n','g','h','a','m','\o'};
4. **Assigning character by character without size**
   - We can assign character by character without size with the NULL character at the end. The size of the string is determined by the compiler automatically.

     char str[ ] = { 'N','o','t','t','i','n','g','h','a','m','\o'};

Note: When a sequence of characters enclosed in the double quotation marks is encountered by the compiler, a null character ‘\o’ is appended at the end of the string by default.

String Initialization cont…

- Memory presentation



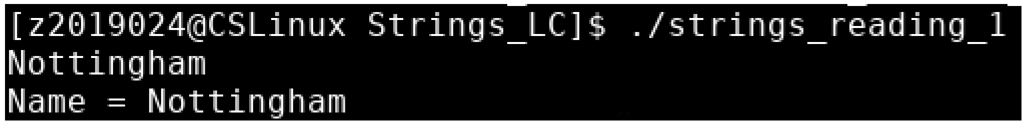
Note: After declaration, if we want to assign some other text to the string, we have to assign it one by one or use built-in strcpy() function because the direct assignment of string literal to character array is only possible in declaration.

Source: https://www.geeksforgeeks.org/

Reading a string

- The C language does not provide an inbuilt data type for strings but it has an access specifier "%s" which can be used to print and read strings directly.

```c
#include<stdio.h>

int main(){
  char name[25];
  scanf("%s", name);
  printf("Name = %s\n", name);
  return(0);
}
```



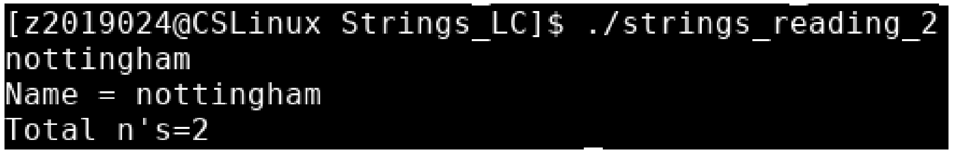
```
[z2019024@CSLinux Strings_LC]$ ./strings_reading_1
Nottingham
Name = Nottingham
```

%s reads a string into a character array given the array name or start address. It ends the string with '\0'

Reading a string cont…

- **Reading a string character-by-character**

```c
#include<stdio.h>

int main(){
  int i, count=0;
  char name[25];
  scanf("%s", name);
  printf("Name = %s\n", name);
  for(i=0;name[i]!='\0';i++)
    if(name[i]=='n')count++;
  printf("Total n's=%d\n", count);
  return(0);
}
```



```
[z2019024@CSLinux Strings_LC]$ ./strings_reading_2
nottingham
Name = nottingham
Total n's=2
```

Note that character strings read in %s format end with '\o'

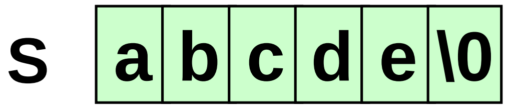
Read the string character-by-character

String array vs string pointer

- **String array**

char s[ ] = “abcde”;
≡ char s[ ] = **{‘a’,’b’,’c’,’d’,’e’,’\o’};**

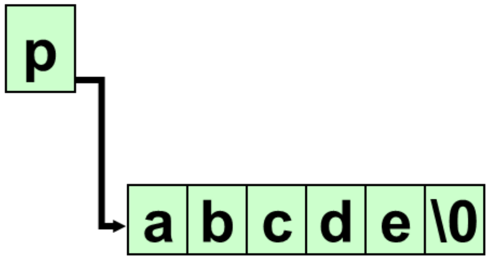
Note: The compiler allocates 6 bytes of memory for the array s which are initialized with the 6 characters



- **String pointer**

char *p = “abcde”;

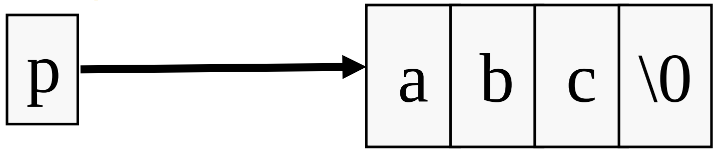
Note: The compiler allocates space for p, puts the string constant “abcde” in memory somewhere else, initializes p with the base address of the string constant



String constant

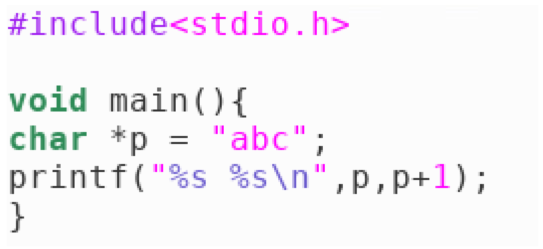
- A string constant is treated as a pointer to the string.
- Its value is the base address of the string

char *p = “abc”;

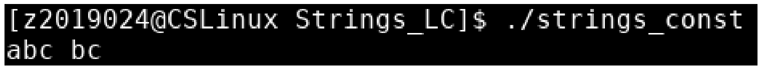


printf (“%s %s\n”,p,p+1);

Output: abc bc



```
#include<stdio.h>

void main(){
char *p = "abc";
printf("%s %s\n",p,p+1);
}
```



```
[z2019024@CSLinux Strings_LC]$ ./strings_const
abc bc
```

String handling functions

- String-Conversion Functions
  - String conversation functions are inside the *stdlib.h* header file. So you need to include this header file in your program to use these functions.
  - **strtod():** converts a string to double
  - **strtol():** converts a string to long
  - **strtoul():** converts a string to unsigned long
  - **atof():** Converts a string to float
  - **atol():** Converts a string to long integer

String handling functions cont...

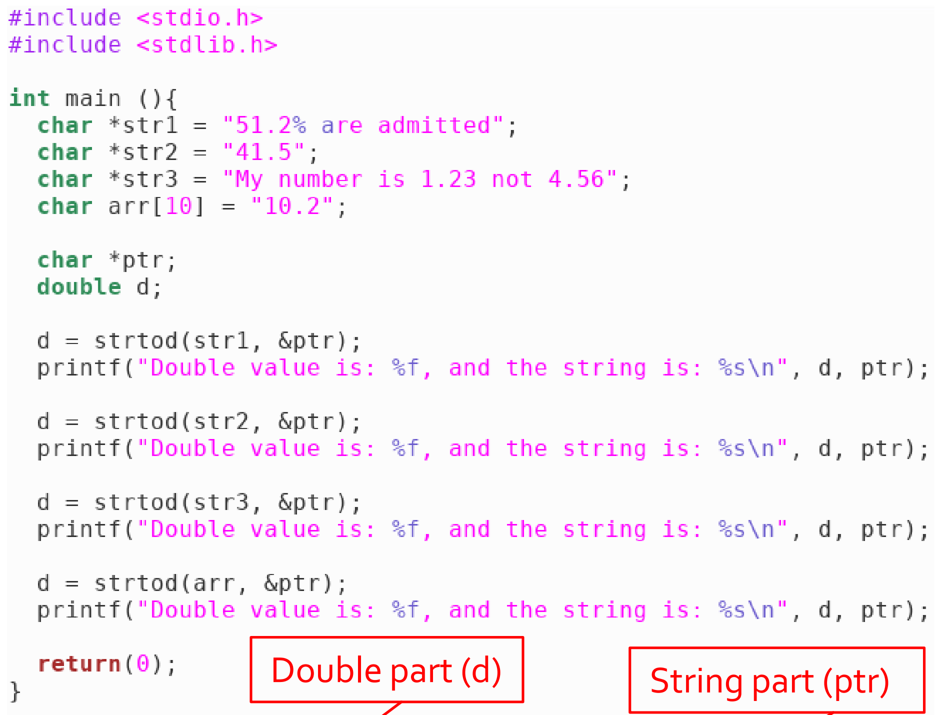
- **strtod()**
  - This function separates double value from a string.
  - The string must begins with a valid floating point number.
  - The pointer receives the memory address of the character after floating point value.
  - On error, point to the beginning of the string.
  - Follow the same rules for the **strtol()** and **strtoul()** functions.
  - Syntax:

It is a pointer to a charter pointer and used to store the pointer to the first character after the numeric value.

It is a string to be converted into a double precision floating-point number.

double strtod(const char *str, char **endptr)

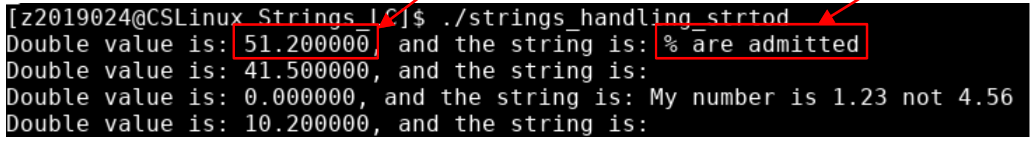
Returns the converted double-precision floating-point number. If the input string is not a valid, it returns 0.



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

int main (){
  char *str1 = "51.2% are admitted";
  char *str2 = "41.5";
  char *str3 = "My number is 1.23 not 4.56";
  char arr[10] = "10.2";

  char *ptr;
  double d;

  d = strtod(str1, &ptr);
  printf("Double value is: %f, and the string is: %s\n", d, ptr);

  d = strtod(str2, &ptr);
  printf("Double value is: %f, and the string is: %s\n", d, ptr);

  d = strtod(str3, &ptr);
  printf("Double value is: %f, and the string is: %s\n", d, ptr);

  d = strtod(arr, &ptr);
  printf("Double value is: %f, and the string is: %s\n", d, ptr);

  return(0);
}
```

Double part (d)

String part (ptr)



```
[z2019024@CSLinux Strings_LC]$ ./strings_handling_strtod
Double value is: 51.200000, and the string is: % are admitted
Double value is: 41.500000, and the string is:
Double value is: 0.000000, and the string is: My number is 1.23 not 4.56
Double value is: 10.200000, and the string is:
```

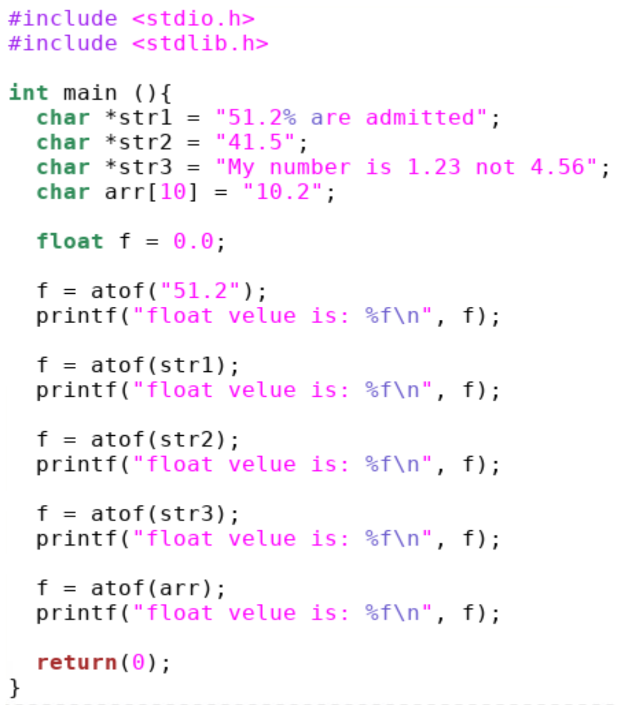
String handling functions cont…

- **atof()**
  - Converts string to float.
  - The string must begins with or will entirely be a valid floating point number.
  - On error, returns zero value.
  - Follow the same rules for the **atol()** function.
  - Syntax:

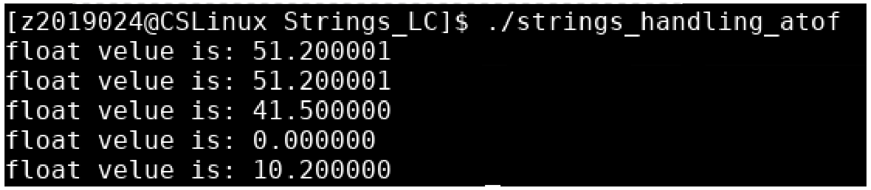
It is a pointer to a null-terminated sting, which represent the a floating point number.

double atof(const char *str)

Returns the converted double-precision floating-point number. If the input string is not a valid, it returns 0.



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

int main (){
  char *str1 = "51.2% are admitted";
  char *str2 = "41.5";
  char *str3 = "My number is 1.23 not 4.56";
  char arr[10] = "10.2";

  float f = 0.0;

  f = atof("51.2");
  printf("float velue is: %f\n", f);

  f = atof(str1);
  printf("float velue is: %f\n", f);

  f = atof(str2);
  printf("float velue is: %f\n", f);

  f = atof(str3);
  printf("float velue is: %f\n", f);

  f = atof(arr);
  printf("float velue is: %f\n", f);

  return(0);
}
```



```
[z2019024@CSLinux Strings_LC]$ ./strings_handling_atof
float velue is: 51.200001
float velue is: 51.200001
float velue is: 41.500000
float velue is: 0.000000
float velue is: 10.200000
```

String handling functions cont…

- Standard string input and output functions
  - String input and output functions belong to *stdio.h* header file. So when will use these functions in your program you need to include this header file.
  - String input functions
    - **scanf():** Input a string from a standard keyboard
    - **sscanf():** Input a string from another string
    - **gets():** Input a string from standard keyboard with blank spaces
    - **fgets():** Input a string from standard keyboard/file line-by-line
  - String output functions
    - **printf():** Print a string to a standard display
    - **fprintf():** Print a string to a standard display/file
    - **sprintf():** Write a string to another string
    - **puts():** Print a string to a standard display and add \n at the end of the string

String handling functions cont…

- **scanf()**
  - Input a string from a standard keyboard.
  - Input a string until blank space or newline encounters.
  - This function can input a string with blank space using scanset.
  - Syntax:

Returns an integer value which indicates the number of input items successfully matched and assigned. If the input does not match the format specifiers, or if the end of the input stream is reached before any matches are made, scanf returns EOF.

The format string specifies the type of input expected and is composed of conversion specifications starting with a % character.

These are the additional arguments corresponding to the format specifiers. Each argument must be a pointer to a variable where the parsed input will be stored.

```
int scanf(const char *format, ...);
```

```
#include <stdio.h>

int main (){
   char str[20];
   scanf("%s", str);

   printf("%s\n", str);

   return(0);
}
```

Character array to store the input string.

Format specifier. %s for sting input.

```
[z2019024@CSLinux Strings_LC]$ ./strings_handling_scanf
Nottingham
Nottingham
[z2019024@CSLinux Strings_LC]$ ./strings_handling_scanf
University of Nottingham Ningbo China
University
```

```
#include <stdio.h>

int main (){
   char str[100];
   scanf("%[^\n]s", str);

   printf("%s\n", str);

   return(0);
}
```

scanset

```
[z2019024@CSLinux Strings_LC]$ ./strings_handling_scanf_scanset
Nottingham
Nottingham
[z2019024@CSLinux Strings_LC]$ ./strings_handling_scanf_scanset
University of Nottingham Ningbo China
University of Nottingham Ningbo China
```

The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

String handling functions cont…

- **sscanf()**
  - Input a string from another string e.g. array, instead of keyboard input.
  - Syntax:

Returns the number of input items successfully matched and assigned.If the input does not match the format string, the function returns EOF or the number of successfully matched and assigned items up to that point.

The input string from which to read.

A format string that specifies how to interpret the input string.

Additional arguments pointing to variables where the extracted values will be stored.

```
int sscanf(const char *str, const char *format, ...);
```

Input string.

Format specifier(s).

Extracted string(s).

```
#include <stdio.h>

int main(){
  int day, year;
  char weekday[20], month[20];
  char dtm[100] = "Friday October 29 2021";
  sscanf( dtm, "%s %s %d %d", weekday, month, &day, &year);
  printf( "%s %d, %d = %s\n", month, day, year, weekday);
  return(0);
}
```

```
[z2019024@CSLinux Strings_LC]$ ./strings_handling_sscanf
October 29, 2021 = Friday
```

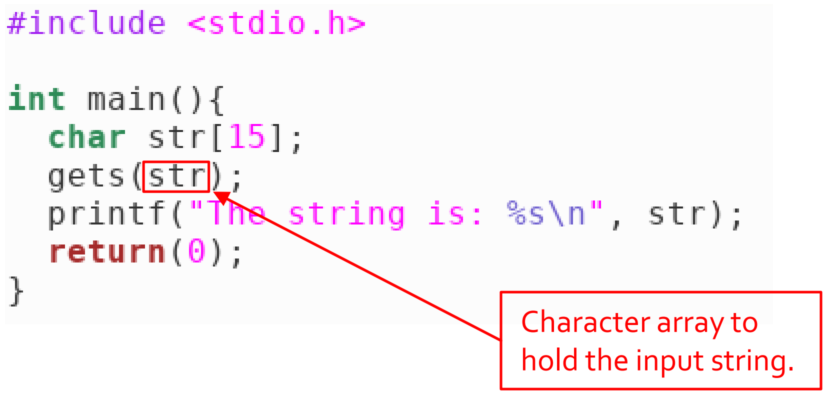
String handling functions cont…

- **gets()**
  - Input a string from standard keyboard with blank spaces.
  - It is not safe to use because it does not check the array bound.
  - It is used to read strings from the user until a newline character is not encountered.
  - Syntax:

Returns the same pointer str on success. On failure or end-of-file condition, it returns NULL.

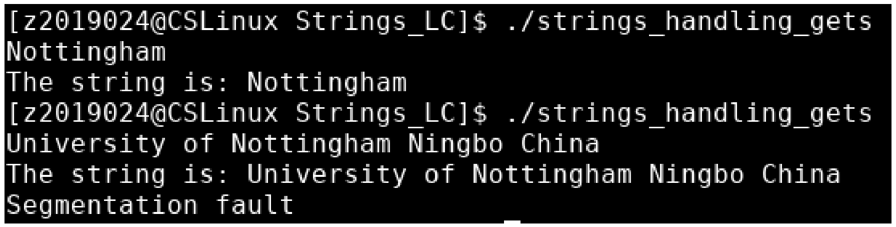
Pointer to the character array where the input string will be stored. It must have enough space to store the input string along with the null terminator.

```
char *gets(char *str);
```



```
#include <stdio.h>

int main(){
  char str[15];
  gets(str);
  printf("The string is: %s\n", str);
  return(0);
}
```

Character array to hold the input string.



```
[z2019024@CSLinux Strings_LC]$ ./strings_handling_gets
Nottingham
The string is: Nottingham
[z2019024@CSLinux Strings_LC]$ ./strings_handling_gets
University of Nottingham Ningbo China
The string is: University of Nottingham Ningbo China
Segmentation fault
```

String handling functions cont…

- **fgets()**
  - Input a string from standard keyboard/file line-by-line.
  - It follows some parameters such as Maximum length, buffer, and input device reference.
  - It is safe to use because it checks the array bound.
  - It keeps on reading until a new line character is encountered or the maximum limit of the character array.
  - Syntax:

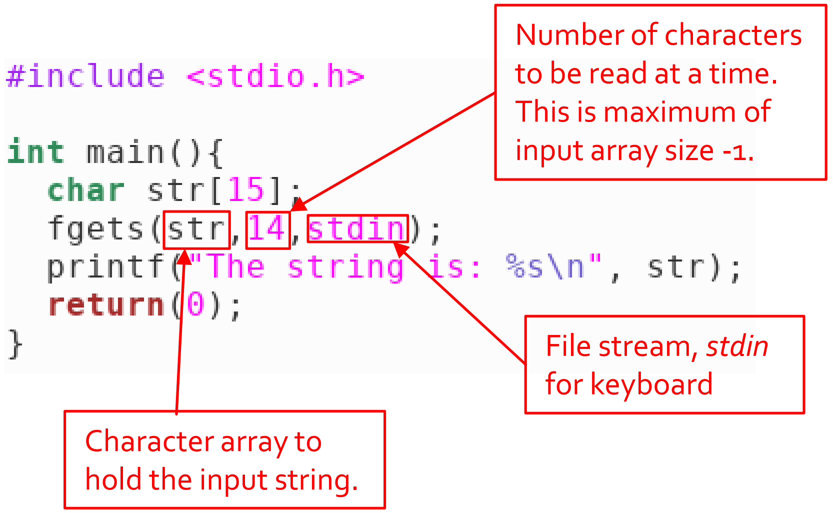
On success, fgets returns the same pointer str that was passed in, which now contains the string that was read. If an error occurs, or if end-of-file is reached and no characters were read, fgets returns NULL.

A pointer to an array of characters where the read string will be stored. This array should be large enough to hold the string, including the terminating null character.

The maximum number of characters to read, including the terminating null character. fgets will read up to n-1 characters, leaving room for the null character.

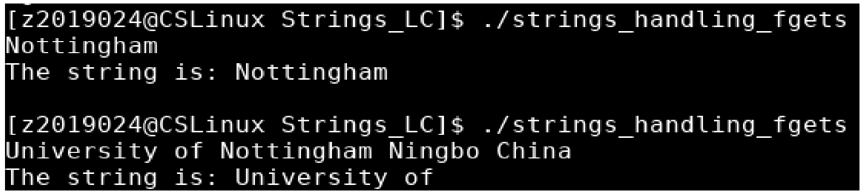
A pointer to a FILE object that specifies the input stream to read from. This can be a file pointer obtained from functions like fopen, or it can be stdin for standard input.

```
char *fgets(char *str, int n, FILE *stream);
```



```
#include <stdio.h>

int main(){
   char str[15];
   fgets(str,14,stdin);
   printf("The string is: %s\n", str);
   return(0);
}
```

Number of characters to be read at a time. This is maximum of input array size -1.
File stream, *stdin* for keyboard
Character array to hold the input string.



```
[z2019024@CSLinux Strings_LC]$ ./strings_handling_fgets
Nottingham
The string is: Nottingham

[z2019024@CSLinux Strings_LC]$ ./strings_handling_fgets
University of Nottingham Ningbo China
The string is: University of
```

String handling functions cont...

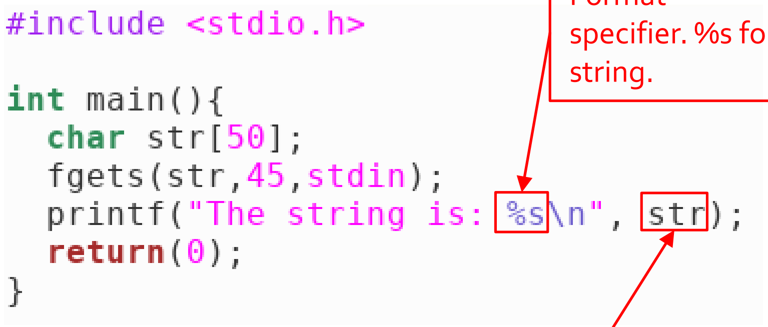
- **printf()**
  - Print a string including blank space to a standard display.
  - Syntax:

Returns the number of characters printed (excluding the null byte used to end the output to strings) if successful. On error, it returns a negative value.

A string that may contain format specifiers like %d, %s, etc., which control the formatting of subsequent arguments.

A variable number of arguments to be formatted and printed according to the format string.

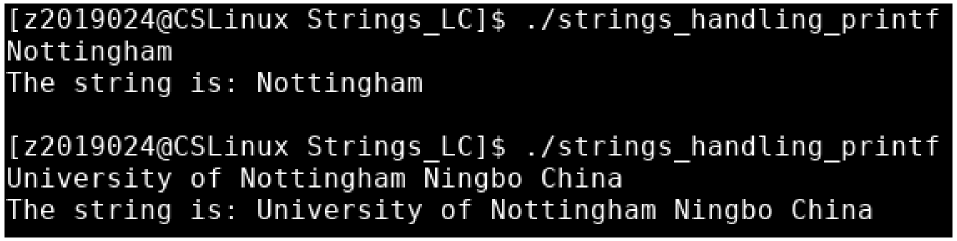
```
int printf(const char *format, ...)
```



```
#include <stdio.h>

int main(){
  char str[50];
  fgets(str,45,stdin);
  printf("The string is: %s\n", str);
  return(0);
}
```

Format specifier. %s for string.

Character array.



```
[z2019024@CSLinux Strings_LC]$ ./strings_handling_printf
Nottingham
The string is: Nottingham

[z2019024@CSLinux Strings_LC]$ ./strings_handling_printf
University of Nottingham Ningbo China
The string is: University of Nottingham Ningbo China
```

String handling functions cont...

- **fprintf()**
  - Print a string to a standard display/file
  - Syntax:

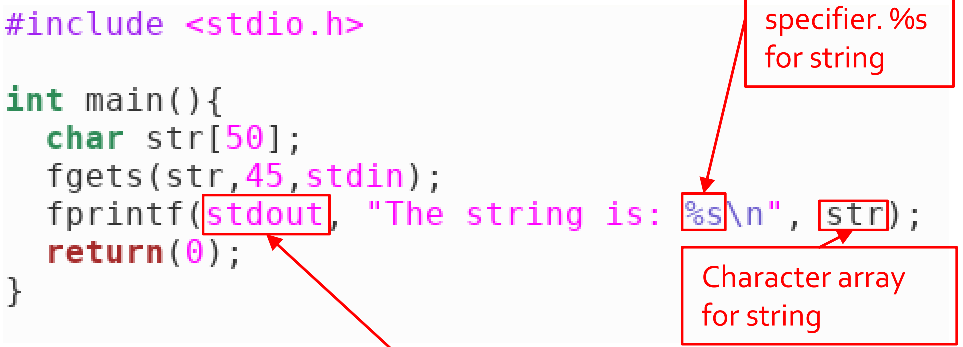
Returns the number of characters written if successful, and a negative value if an error occurs.

A pointer to the FILE object that identifies the stream where the output is to be written. *stdout* for standard output device.

A string that specifies the format of the output. It may contain format specifiers that are replaced by the values specified in the subsequent arguments.

Additional arguments that correspond to the format specifiers in the format string.

int fprintf(FILE *stream, const char *format, ...);

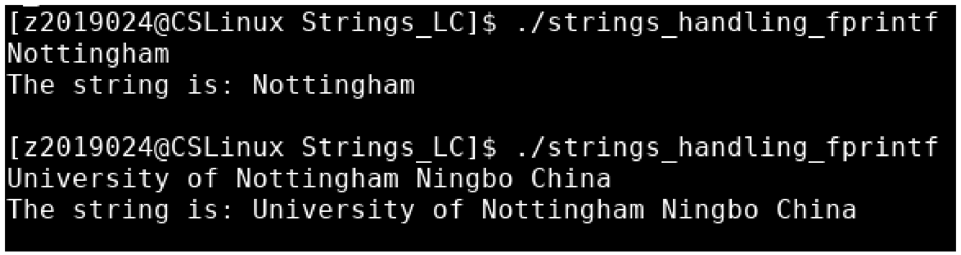


```
#include <stdio.h>

int main(){
  char str[50];
  fgets(str,45,stdin);
  fprintf(stdout, "The string is: %s\n", str);
  return(0);
}
```

Format specifier. %s for string

Character array for string

Output device. *stdout* for standard output.



```
[z2019024@CSLinux Strings_LC]$ ./strings_handling_fprintf
Nottingham
The string is: Nottingham

[z2019024@CSLinux Strings_LC]$ ./strings_handling_fprintf
University of Nottingham Ningbo China
The string is: University of Nottingham Ningbo China
```

String handling functions cont…

- **sprintf()**
  - Writes a string to another string e.g. array, instead of screen.
  - Syntax:

Returns the number of characters written to the string str, excluding the null-terminating character.

A pointer to an array of characters where the resulting string will be stored.

A pointer to a null-terminated string that contains the text to be written to the string str. This string may contain format specifiers that dictate how subsequent arguments are converted and formatted into the resulting string.

Additional arguments that correspond to the format specifiers in the string.

```
int sprintf(char *str, const char *format, ...);
```

```c
#include <stdio.h>

int main (){
   char str[20] = {'\0'};
   sprintf(str, "Hello World!");
   printf("%s\n", str);

   return(0);
}
```

String to be written.

Character array to write the string.

```
[z2019024@CSLinux Strings_LC]$ ./strings_handling_sprintf
Hello World!
```

String handling functions cont…

- **puts()**
  - Print a string to a standard display and add \n at the end of the string.
  - Syntax:

On success, this function returns a non-negative integer and on failure, it returns EOF (End Of File).

A pointer to a null-terminated string that you want to print.

```
int puts(const char *str);
```

```c
#include <stdio.h>

int main (){
   char str[] = "University of Nottingham Ningbo China";
   puts("Hello world!");
   puts(str);
   return(0);
}
```

The string you want to print

```
[z2019024@CSLinux Strings_LC]$ ./strings_handling_puts
Hello world!
University of Nottingham Ningbo China
```

String handling functions cont…

- Basic string operation functions:
  - These functions are inside *string.h* header file. When we use these functions we need to include *string.h* header file in our program.
  - **strlen():** Estimates length of a string.
  - **strcpy() & strncpy():** Copy a source string to a destination string.
  - **strcat() & strncat():** Concatenate two strings.
  - **strcmp() & strncmp():** Compares two strings.
  - **strchr() & strrchr():** Search a character inside a string.
  - **strstr():** Search a string inside another string

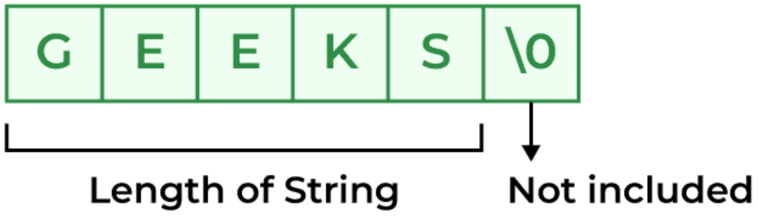
String handling functions cont…

size_t strlen(const char *str)

- This function returns the integral length of the string (*str*) passed.
- strlen() does not count the NULL character '\o'.

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

int main (){
  char str[] = "Nottingham";
  int length = strlen(str);
  printf("Length of string is: %d\n", length);
  return(0);
}
```

```
[z2019024@CSLinux Strings_LC]$ ./strings_handling_strlen
Length of string is: 10
```



Source: https://www.geeksforgeeks.org/

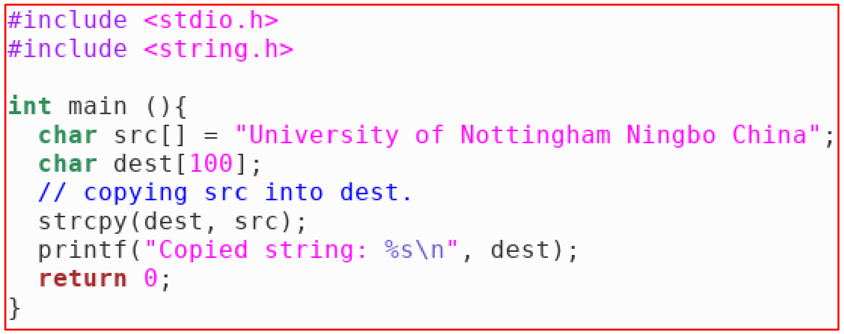
String handling functions cont…

char *strcpy(char *dest, const char *src)

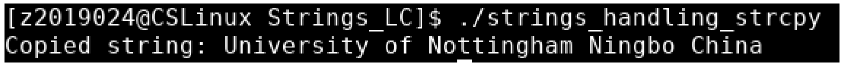
- strcpy() is a C standard library function that copies a string from one location to another.
- The function takes two arguments: a destination buffer (dest) where the copied string will be stored, and a source string (src) that will be copied. The function copies the entire source string, including the null terminator, into the destination buffer.
- Using this function, you can copy the entire string to the destination string. Source strings are not appended to destination strings. As a result, the content of the destination string is replaced by the content of the source string.
- Source strings are not affected. After copying, the source string remains the same.
- In the case of a longer source string (Character Array), strcpy() performs undefined behavior.

char *strncpy(char *dest, const char *src, size_t n):

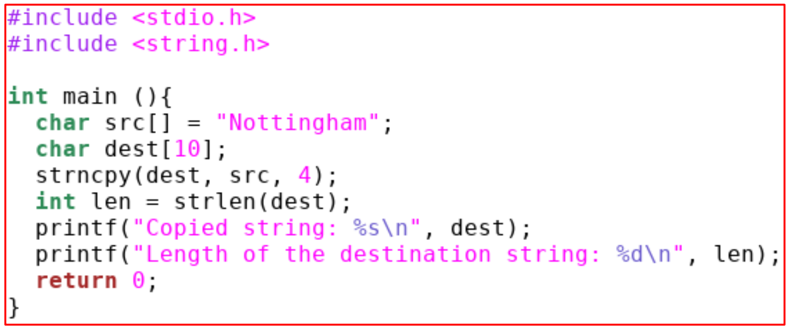
- Copies the first *n* characters of *source* to *destination.*
- If there is no NULL character among the first n character of src, the string placed in dest will not be NULL-terminated.
- If the length of *src* is less than *n*, strncpy() writes additional NULL character to *dest* to ensure that a total of *n* character are written.



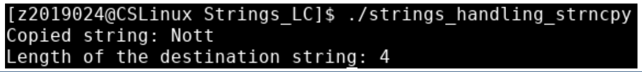
```c
#include <stdio.h>
#include <string.h>

int main (){
  char src[] = "University of Nottingham Ningbo China";
  char dest[100];
  // copying src into dest.
  strcpy(dest, src);
  printf("Copied string: %s\n", dest);
  return 0;
}
```



```
[z2019024@CSLinux Strings_LC]$ ./strings_handling_strcpy
Copied string: University of Nottingham Ningbo China
```



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

int main (){
  char src[] = "Nottingham";
  char dest[10];
  strncpy(dest, src, 4);
  int len = strlen(dest);
  printf("Copied string: %s\n", dest);
  printf("Length of the destination string: %d\n", len);
  return 0;
}
```



```
[z2019024@CSLinux Strings_LC]$ ./strings_handling_strncpy
Copied string: Nott
Length of the destination string: 4
```

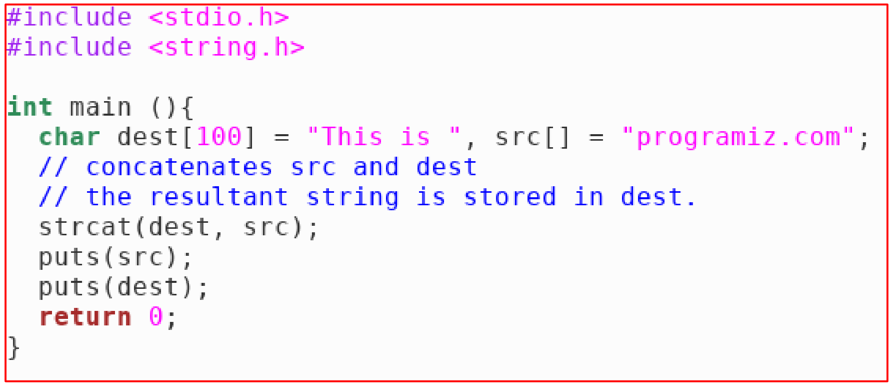
String handling functions cont…

char *strcat(char *dest, const char *src)

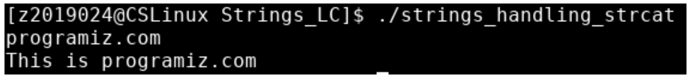
- It will append a copy of the source string in the destination string. plus a terminating Null character.
- The initial character of the source string overwrites the Null-character present at the end of the destination string.
- The behavior is undefined if the strings overlap and the *dest* array is not large enough to append the contents of *src*.

char *strncat(char *dest, const char *src, size_t n)

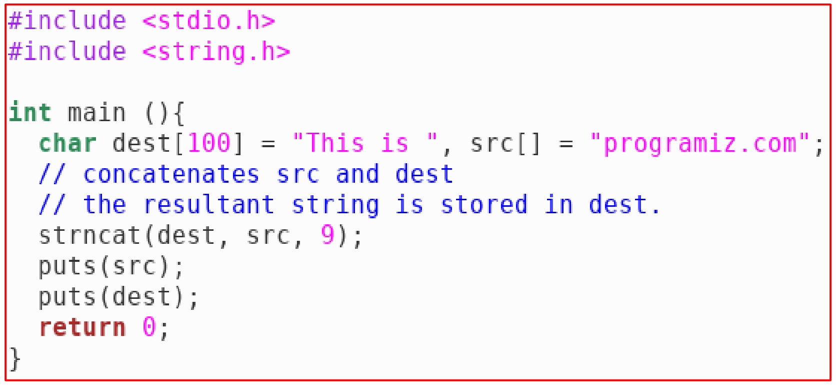
- This function appends not more than n characters from the source string to the end of the destination string plus a terminating Null-character.
- The initial character of the source string overwrites the Null-character present at the end of the destination string.
- Thus, the length of the string(dest) becomes strlen(dest)+n.
- But, if the length of the string(src) is less than n, only the content up to the terminating null-character is copied and the length of the string(dest) becomes strlen(src) + strlen(dest).
- The behavior is undefined if the strings overlap and the *dest* array is not large enough to append the contents of *src*.



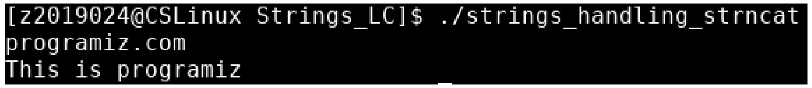
```c
#include <stdio.h>
#include <string.h>

int main (){
  char dest[100] = "This is ", src[] = "programiz.com";
  // concatenates src and dest
  // the resultant string is stored in dest.
  strcat(dest, src);
  puts(src);
  puts(dest);
  return 0;
}
```



```
[z2019024@CSLinux Strings_LC]$ ./strings_handling_strcat
programiz.com
This is programiz.com
```



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

int main (){
  char dest[100] = "This is ", src[] = "programiz.com";
  // concatenates src and dest
  // the resultant string is stored in dest.
  strncat(dest, src, 9);
  puts(src);
  puts(dest);
  return 0;
}
```



```
[z2019024@CSLinux Strings_LC]$ ./strings_handling_strncat
programiz.com
This is programiz
```

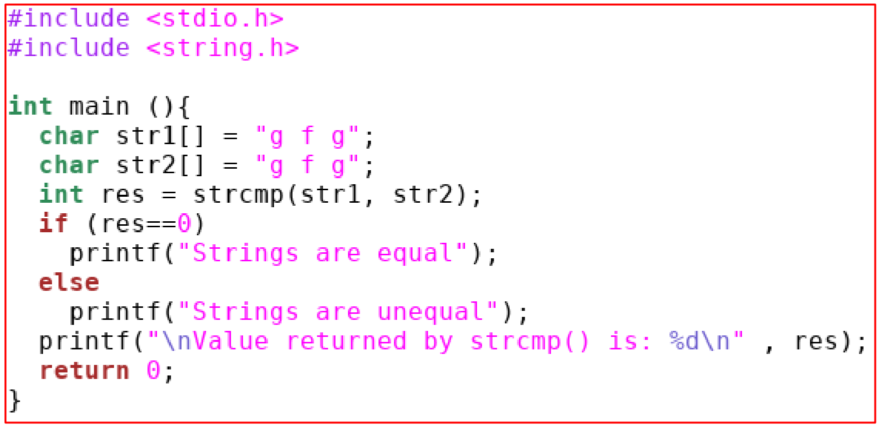
String handling functions cont…

strcmp(const char *str1, const char *str2)

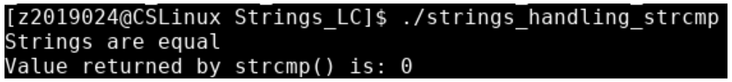
- This function takes two strings (array of characters) as arguments, compares these two strings lexicographically.
- Returns zero if it is the same string.

strncmp(const char *str1, const char *str2, size_t n):

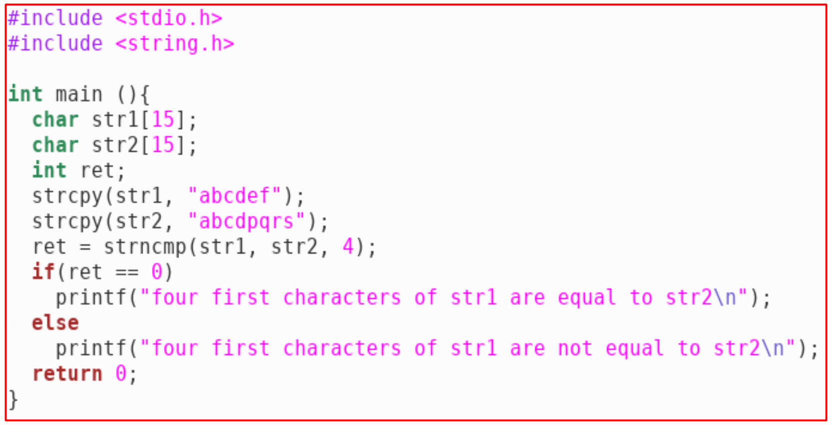
- This function lexicographically compares two strings upto *n* characters.
- Returns zero if the first *n* characters are the same.



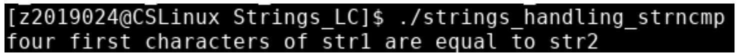
```
#include <stdio.h>
#include <string.h>

int main (){
  char str1[] = "g f g";
  char str2[] = "g f g";
  int res = strcmp(str1, str2);
  if (res==0)
    printf("Strings are equal");
  else
    printf("Strings are unequal");
  printf("\nValue returned by strcmp() is: %d\n" , res);
  return 0;
}
```



```
[z2019024@CSLinux Strings_LC]$ ./strings_handling_strcmp
Strings are equal
Value returned by strcmp() is: 0
```



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

int main (){
  char str1[15];
  char str2[15];
  int ret;
  strcpy(str1, "abcdef");
  strcpy(str2, "abcdpqrs");
  ret = strncmp(str1, str2, 4);
  if(ret == 0)
    printf("four first characters of str1 are equal to str2\n");
  else
    printf("four first characters of str1 are not equal to str2\n");
  return 0;
}
```



```
[z2019024@CSLinux Strings_LC]$ ./strings_handling_strncmp
four first characters of str1 are equal to str2
```

String handling functions cont...

char *strchr(const char *str, char c)

- This function searches for the first occurrence of the character *char* (an unsigned char) in the string pointed to by the argument *str*.
- This returns a pointer to the first occurrence of the character *char* in the string *str*, or NULL if the character is not found.

char *strrchr(const char *str, char c)

- This function searches for the last occurrence of the character *char* (an unsigned char) in the string pointed to, by the argument *str*.
- This function returns a pointer to the last occurrence of character *char* in *str*. If the value is not found, the function returns a null pointer.

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

int main (){
  const char str[] = "http://www.tutorialspoint.com";
  const char ch = '.';
  char *ret;
  ret = strchr(str, ch);
  printf("String after |%c| is - |%s|\n", ch, ret);
  return 0;
}
```

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

int main (){
  const char str[] = "http://www.tutorialspoint.com";
  const char ch = '.';
  char *ret;
  ret = strrchr(str, ch);
  printf("String after |%c| is - |%s|\n", ch, ret);
  return 0;
}
```

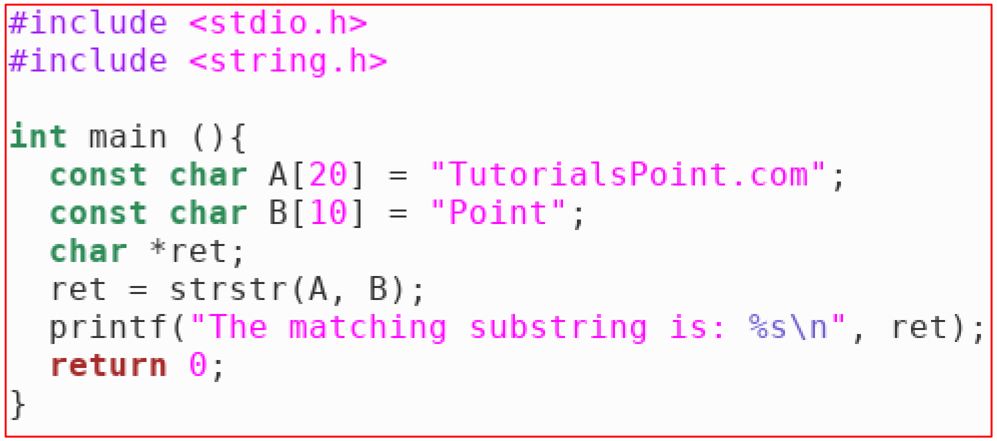
```
[z2019024@CSLinux Strings_LC]$ ./strings_handling_strchr
String after |.| is - |.tutorialspoint.com|
```

```
[z2019024@CSLinux Strings_LC]$ ./strings_handling_strrchr
String after |.| is - |.com|
```

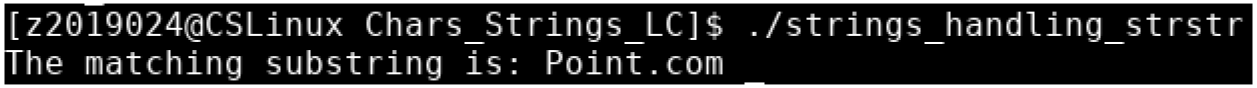
String handling functions cont…

char *strstr(const char *A, const char *B)

- This function finds the first occurrence of the substring "B" in the string "A". The terminating '\o' characters are not compared.
- This function returns a pointer to the first occurrence in A of any of the entire sequence of characters specified in B, or a null pointer if the sequence is not present in A.



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

int main (){
   const char A[20] = "TutorialsPoint.com";
   const char B[10] = "Point";
   char *ret;
   ret = strstr(A, B);
   printf("The matching substring is: %s\n", ret);
   return 0;
}
```



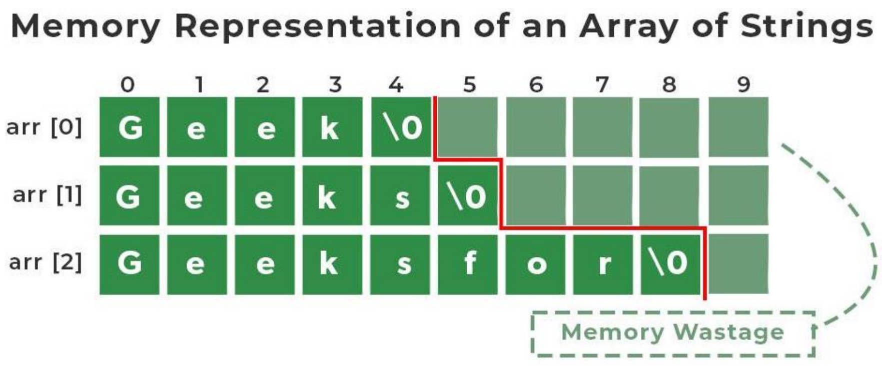
```
[z2019024@CSLinux Chars_Strings_LC]$ ./strings_handling_strstr
The matching substring is: Point.com
```

Array of strings

- In C programming, String is a 1-D array of characters and is defined as an array of characters. But an array of strings in C is a two-dimensional array of character types. Each String is terminated with a null character (\0). It is an application of a 2d array.

```c
#include <stdio.h>

int main (){
  char arr[3][10] = {"Geek",
                     "Geeks", "Geekfor"};
  printf("String array Elements are:\n");

  for (int i = 0; i < 3; i++)
  {
    printf("%s\n", arr[i]);
  }
  return 0;
}
```

```
[z2019024@CSLinux Strings_LC]$ ./array_of_strings_arr
String array Elements are:
Geek
Geeks
Geekfor
```



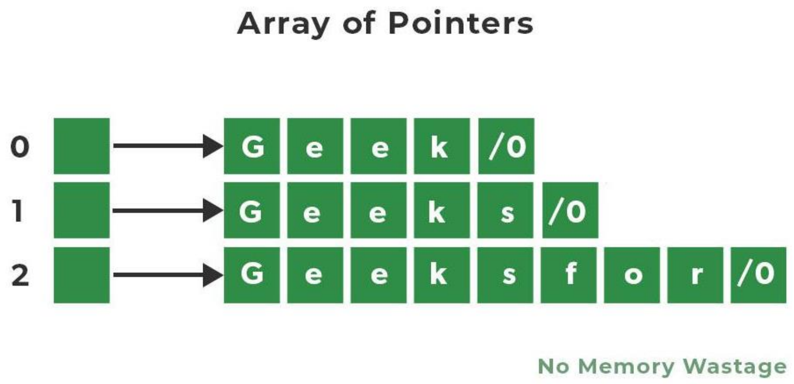
Source: https://www.geeksforgeeks.org/

Array of strings cont…

- In C, we can use an Array of pointers. Instead of having a 2-Dimensional character array, we can have a single-dimensional array of Pointers. Here pointer to the first character of the string literal is stored.

```c
#include <stdio.h>

int main (){
  char *arr[] = {"Geek", "Geeks", "Geekfor"};
  printf("String array Elements are:\n");

  for (int i = 0; i < 3; i++)
  {
    printf("%s\n", arr[i]);
  }
  return 0;
}
```

```
[z2019024@CSLinux Strings_LC]$ ./array_of_strings_ptr
String array Elements are:
Geek
Geeks
Geekfor
```



Source: https://www.geeksforgeeks.org/

Conclusion

- String is a 1-d array of type char.
- A string in C is terminated by '\0' (null character).
- C provides string input and output functions as well as basic string operation functions.
- Pointer can be used to build an array of strings without wastage of memory.

Thank you!