Strings

Fundamentals of Strings and Characters

- Characters are the fundamental building blocks of source programs.
- Every program is composed of a sequence of characters that—when grouped together meaningfully—is interpreted by the computer as a series of instructions used to accomplish a task.
- The value of a character constant is the integer value of the character in the machine’s character set.

Contd…

- Your computer’s memory stores all data in numeric form.
- There is no direct way to store characters.
- However, a numeric code exists for each character.
- This is called the ASCII code or the ASCII character set. (ASCII stands for American Standard Code for Information Interchange.)
- The code assigns values between 0 and 255 for uppercase and lowercase letters, numeric digits, punctuation marks, and other symbols.

Contd…

- For example, `'z'` represents the integer value of `z`, and `'\n'` the integer value of newline (122 and 10 in ASCII, respectively).
- Variables of type char can hold only a single character, so they have limited usefulness.
- A string is a series of characters treated as a single unit.
- A string may include letters, digits and various special characters such as +, -, *, / and $.
- String literals, or string constants, in C are written in double quotation marks.

Contd…

- A string in C is an array of characters ending in the null character (`'\0'`).
- A string is accessed via a *pointer* to the first character in the string.
- The value of a string is the address of its first character.
- Thus, in C, it’s appropriate to say that a string is a pointer—in fact, a pointer to the string’s first character.
- In this sense, strings are like arrays, because an array is also a pointer to its first element.
- A *character array* or a *variable of type* ***`char *`*** can be initialized with a string in a definition.

Contd…

- To hold a string of six characters, for example, you need to declare an array of type char with seven elements.
- Arrays of type char are declared like arrays of other data types.
- For example, the statement char string[10]; declares a 10-element array of type char.
- This array could be used to hold a string of nine or fewer characters.

Contd…

- “But wait,” you might be saying. “It’s a 10-element array, so why can it hold only nine characters?
- In C, a string is defined as a sequence of characters ending with the null character.
- The null character is a special character represented by \0.
- Although it’s represented by two characters (backslash and zero), the null character is interpreted as a single character and has the ASCII value of 0.

Contd…

- The definitions

  -
    ```
    char color[5] = "blue";
    char *colorPtr = "blue";
    ```

  each initialize a variable to the string `"blue"`.
- The first definition creates a 5-element array `color` containing the characters `'b'`, `'l'`, `'u'`, `'e'` and `'\0'`.
- The second definition creates pointer variable `colorPtr` that points to the string `"blue"` somewhere in memory.

Contd…

- Char *message;
- This statement declares a pointer to a variable of type char named message.
- It doesn’t point to anything now, but what if you changed the pointer declaration to read:
- char *message = “Great Ghost!”;

Contd…

- When this statement executes, the string Great Ghost! (with a terminating null character) is stored somewhere in memory, and the pointer message is initialized to point to the first character of the string.
- Don’t worry where in memory the string is stored; it’s handled automatically by the compiler.
- Once defined, message is a pointer to the string and can be used as such.

Contd…

- The preceding declaration/initialization is equivalent to the following, and the two notations *message and message[] also are equivalent; they both mean “a pointer to.”
char message[] = “Great Ghost!”;

Contd…

- The preceding array definition could also have been written
  - `char color[] = { 'b', 'l', 'u', 'e', '\0' };`
- When defining a character array to contain a string, the array must be large enough to store the string *and* its terminating null character.
- The preceding definition automatically determines the size of the array based on the number of initializers in the initializer list.

Contd…

- A string can be stored in an array using `scanf`.
- For example, the following statement stores a string in character array `word[20]`:
  - `scanf( "%s", word );`
- The string entered by the user is stored in `word`.
- Variable `word` is an array, which is, of course, a pointer, so the `&` is not needed with argument `word`.

Contd…

- How does scanf() decide where the string begins and ends?
- The beginning is the first non-whitespace character encountered.
- The end can be specified in one of two ways.
- If you use %s in the format string, the string runs up to (but not including) the next whitespace character (space, tab, or newline).
- If you use %ns (where n is an integer constant that specifies field width), scanf() accepts the next n characters or up to the next whitespace character, whichever comes first.

Contd…

- You can read in multiple strings with scanf() by including more than one %s in the format string.
- For each %s in the format string, scanf() uses the preceding rules to find the requested number of strings in the input.
- For example scanf(“%s%s%s”, s1, s2, s3); If in response to this statement you enter January February March, January is assigned to the string s1, February is assigned to s2, and March to s3.

Contd…

- What about using the field-width specifier?
- If you execute the statement scanf(“%3s%3s%3s”, s1, s2, s3); and in response you enter September, Sep is assigned to s1, tem is assigned to s2, and ber is assigned to s3.

Contd…

- What if you enter fewer or more strings than the scanf() function expects?
- If you enter fewer strings, scanf() continues to look for the missing strings, and the program doesn't continue until they're entered.
- For example, if in response to the statement scanf("%s%s%s", s1, s2, s3); you enter January February, the program sits and waits for the third string specified in the scanf() format string.

Contd…

- If you enter more strings than requested, the unmatched strings remain pending (waiting in the keyboard buffer) and are read by any subsequent scanf() or other input statements.
- For example, if in response to the statements scanf(“%s%s”, s1, s2); scanf(“%s”, s3); you enter January February March, the result is that January is assigned to the string s1 and February is assigned to s2 in the first scanf() call.
- March is then automatically carried over and assigned to s3 in the second scanf() call.

Conversion Specifiers

- %s - Scan a character string.
- The scan terminates at whitespace.
- A null character is stored at the end of the string, which means that the buffer supplied must be at least one character longer than the specified input length.

Contd…

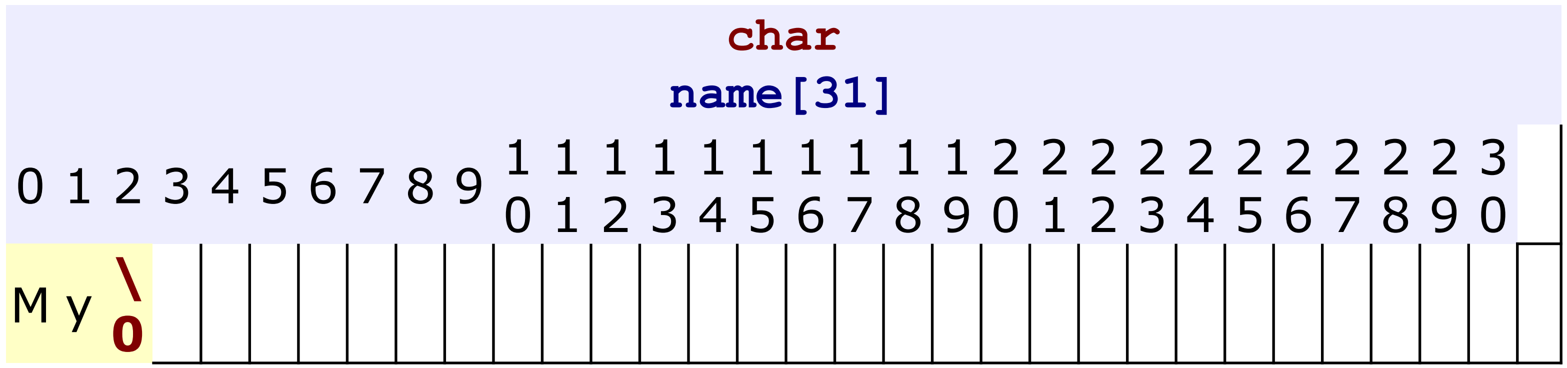
- For example,

```
char name[31];
scanf("%s", name);
```

Input : My name is Arnold

It stops accepting input after the character **y** and stores the following:

Contd…



char
name[31]
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
M y \0

Contd…

- The characters **' name is Arnold'** remain in the input buffer.
- A qualifier on the conversion specifier limits the number of characters accepted. For instance, %10s reads no more than 10 characters.
- char name[31];
- scanf("%10s", name);

Input : Schwartzenegger

Contd…

char name[31]																														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
S	c	h	w	a	r	t	z	e	n	\0																				

Contd…

- It stops after the character n.
- By specifying the maximum number of characters to be read at less than 31, we ensure that **scanf()**does not exceed the memory allocated for the string.

Conversion Specifier

- **%[ ]**
- For example, the **%[^\n]** conversion specifier:
- reads all characters until the newline (**'\n'**)
- stores the characters read in the **char** array identified by the corresponding argument
- stores the null terminator in the **char** array after accepting the last character
- leaves the delimiting character (here, **'\n'**) in the input buffer

Contd…

- For example,

```
char name[31];
scanf("%[^\n]", name);
```

- Input: My name is Arnold

Contd…

- It accepts full line and stores

char name[31]																														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
M	y		n	a	m	e		i	s		A	r	n	o	l	d	**\0**													

Contd…

- A qualifier on this conversion specifier before the opening bracket limits the number of characters accepted.
- For instance, **%10[^\n]** reads no more than 10 characters.

Contd…

- char name[31];
- scanf("%10[^\n]", name);
- Input:
  My name is Arnold

Contd…

char name[31]																														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
M	y		n	a	m	e		i	s	**\0**																				

Contd…

- **%[ ]**, like **%s**, ignores any leading whitespace characters.
- My name is Arnold

char name[31]																														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
M	y		n	a	m	e		i	s	\0																				

Fscanf

- The C library function **int fscanf(FILE *stream, const char *format, ...)**reads formatted input from a stream.

Contd....

```
#include <stdio.h>
#include <stdlib.h>
int main() {
char str1[10], str2[10], str3[10];
int year;
FILE * fp;
fp = fopen ("file.txt", "w+");
fputs("We are in 2017", fp);
rewind(fp);
fscanf(fp, "%s %s %s %d", str1, str2, str3, &year);
printf("Read String1 |%s|\n", str1 );
printf("Read String2 |%s|\n", str2 );
printf("Read String3 |%s|\n", str3 );
printf("Read Integer |%d|\n", year );
fclose(fp);
return(0);
}
```

Ouput

```
Read String1 |We|
Read String2 |are|
Read String3 |in|
Read Integer |2017|
```

Displaying Strings and Characters

- If your program uses string data, it probably needs to display the data on the screen at some time.
- String display is usually done with either the puts() function or the printf() function.

Puts function

- The puts() function puts a string on-screen—hence its name.
- A pointer to the string to be displayed is the only argument puts() takes.
- The puts() function automatically inserts a new line character at the end of each string it displays, so each subsequent string displayed with puts() is on its own line.

Contd…

```
// Displaying Lines
// puts.c
#include <stdio.h>
int main(void) {
const char name[31] = "My name is Arnold";
puts(name);
return 0;
}
```

printf function

- When printf() encounters a %s in its format string, the function matches the %s with the corresponding argument in its argument list.
- For a string, this argument must be a pointer to the string that you want displayed.
- The printf() function displays the string onscreen, stopping when it reaches the string’s terminating null character.
- For example: char *str = “A message to display”; printf(“%s”, str);

Fputs function

- **fputs()** writes a null-terminated string to a file. The prototype for **fputs()** is:
- int fputs(char **str*, FILE **fp*); ***str*** receives the address of the string to be written
  and ***fp*** receives the address of the **FILE** object.

gets and puts function

```
#include <stdio.h>
int main()
{
        char name[30];
        printf("Enter name: ");
        gets(name);
        printf("Name: ");
        puts(name);
        return 0;
}
```

Output

```
Enter name: Tom Hanks
Name: Tom Hanks
```

String handling

- You need to often manipulate strings according to the need of a problem.
- Most, if not all, of the string manipulation can be done manually but, this makes programming complex and large.
- To solve this, C supports a large number of string handling functions in the standard library"string.h".

Contd…

Function	Work of Function
strlen()	Calculates the length of string
strcpy()	Copies a string to another string
strcat()	Concatenates(joins) two strings
strcmp()	Compares two string

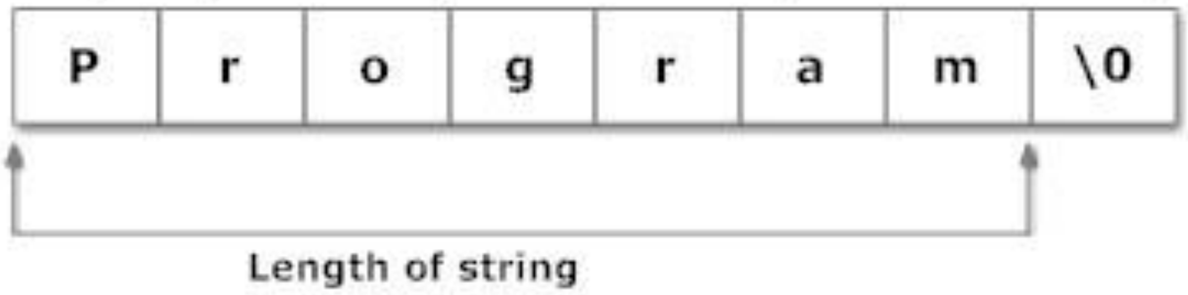
strlen()

- The function takes a single argument, i.e, the string variable whose length is to be found, and returns the length of the string passed excluding the null character.

strlen()

```
char c[]={'P', 'r', 'o', 'g', 'r', 'a', 'm', '\0'};
temp=strlen(c);
```

Then, temp will be equal to 7 because, null character '\0' is not counted.



Example

```
#include <stdio.h>
#include <string.h>
int main() {
char a[20]="Program";
char b[20]={'P','r','o','g','r','a','m','\0'};
char c[20];
printf("Enter string: ");
gets(c);
printf("Length of string a = %d \n",strlen(a));
printf("Length of string b = %d \n",strlen(b));
printf("Length of string c = %d \n",strlen(c));
return 0;
}
```

Output

```
Enter string: String
Length of string a = 7
Length of string b = 7
Length of string c = 6
```

strcpy()

- The strcpy() function copies the string pointed by source (including the null character) to the character array destination.
- This function returns character array destination.
- The strcpy() function is defined in string.h header file.

Example

```
#include <stdio.h>
#include <string.h>
int main()
{
        char str1[10]= "awesome";
        char str2[10];
        char str3[10];
        strcpy(str2, str1);
        strcpy(str3, "well");
        puts(str2);
        puts(str3);
        return 0;
}
```

Note

It is important to note that, the destination array should be large enough otherwise it may result in undefined behavior.

strcat()

- In C programming, strcat() concatenates (joins) two strings.
- It takes two arguments, i.e, two strings or character arrays, and stores the resultant concatenated string in the first string specified in the argument.
- The pointer to the resultant string is passed as a return value.

strcat()

```
#include <stdio.h>
#include <string.h>
int main()
{
        char str1[] = "This is", str2[] = " C programming";
        strcat(str1,str2);
        puts(str1);
        puts(str2);
        return 0;
}
```

strcmp()

- The strcmp() function compares two strings and returns 0 if both strings are identical.
- It takes two strings and return an integer.
- It compares two strings character by character.
- If the first character of two strings are equal, next character of two strings are compared.
- This continues until the corresponding characters of two strings are different or a null character '\0' is reached.

strcmp()

Return Value	Remarks
0	if both strings are identical (equal)
negative	if the ASCII value of first unmatched character is less than second.
positive integer	if the ASCII value of first unmatched character is greater than second.

Example

```
#include <stdio.h>
#include <string.h>
int main()
{
	char str1[] = "abcd", str2[] = "abCd", str3[] ="abcd";
	int result;
	result = strcmp(str1, str2);
	printf("strcmp(str1, str2) = %d\n", result);
	result = strcmp(str1, str3);
	printf("strcmp(str1, str3) = %d\n", result);
	return 0;
}
```

Output

strcmp(str1, str2) = 32

strcmp(str1, str3) = 0

The first unmatched character between string str1 and str2 is third character. The ASCII value of 'c' is 99 and the ASCII value of 'C' is 67. Hence, when strings str1 and str2 are compared, the return value is 32.

When strings str1 and str3 are compared, the result is 0 because both strings are identical.

Input Functions

- The **stdio** library functions for processing input are:

scanf() - input from standard input under format control

fscanf() - input from file under format control

getchar() - character by character input from standard input

fgetc() - character by character input from file

gets_s() - line by line input from standard input

fgets() - line by line input from file

Unformatted Input Functions

- The library functions for processing unformatted input are:
- **getchar()** - character by character input from standard input
- **fgetc()** - character by character input from file
- **gets_s()** - line by line input from standard input
- **fgets()** - line by line input from file

Output functions

The **stdio** library functions for processing output are:

- **printf()** - output to standard output under format control
- **fprintf()** - output to a file under format control
- **putchar()** - character by character output to standard output
- **fputc()** - character by character output to a file
- **puts()** - character string output to standard output
- **fputs()** - character string output to a file

Calculate Length of String without Using strlen() Function

```
#include <stdio.h>
int main()
{
        char s[1000], i;
        printf("Enter a string: ");
        scanf("%s", s);
        for(i = 0; s[i] != '\0'; ++i);
        printf("Length of string: %d", i);
        return 0;
}
```

Find the Frequency of Characters

```c
#include <stdio.h>
int main()
{
		char str[1000], ch;
		int i, frequency = 0;
		printf("Enter a string: ");
		gets(str);
		printf("Enter a character to find the frequency: ");
		scanf("%c",&ch);
		for(i = 0; str[i] != '\0'; ++i)
		{
			if(ch == str[i])
			++frequency;
		}
			printf("Frequency of %c = %d", ch, frequency);
		return 0;
}
```

Program to Check Whether a Character is an Alphabet or not

```
#include <stdio.h>
int main()
{
   char c;
   printf("Enter a character: ");
   scanf("%c",&c);

   if( (c>='a' && c<='z') || (c>='A' && c<='Z'))
      printf("%c is an alphabet.",c);
   else
      printf("%c is not an alphabet.",c);

   return 0;
}
```