FUNCTIONS

QUICK EXERCISE

Write a function that compares two numbers and returns 1 if equal else returns 0.

Use this function to compare first and last elements of the array and print “Equal” or “Not Equal”

SOLUTION

```
int compare (int a, int b)
{
if(a==b)
return 1;
else
return 0;
}
int main()
{
int a[10],n,i,count=0;
scanf("%d", &n);
for(i=0;i<n;i++)
scanf("%d", &a[i]);
if(compare(a[0],a[n-1])==1)
printf(“Equal”);
else
printf(“Not equal”);
return 0;
}
```

PASSING ARRAY TO A FUNCTION

- In the previous program, we individually passed the elements of an array one by one.
- In some cases, we may have to pass an ENTIRE array to the function.
- The function may want to access all elements of the array.
- How can we pass an entire array at once?
- What information does the function need?
  - Address of the first element of the array
- To pass an entire array to the function, we only need to pass the array name.

PASSING ARRAYS TO FUNCTIONS

- Syntax of function definition:
- There are three ways to write formal arguments of a function that takes an array as a parameter:
- Way-1: Formal parameters as a pointer

```
void myFunction(int *arr) { . . . }
```

- Way-2: Formal parameters as a sized array

```
void myFunction(int arr[SIZE]) { . . . }
```

- Way-3: Formal parameters as an unsized array

```
void myFunction(int arr[]) { . . . }
```

QUICK EXERCISE(FILL IN THE BLANKS)

- Program that calculates and returns sum of all array elements.

```
#include <stdio.h>
int sum(___________) //Formal parameters
{
int s=0,i;
//Calculate the sum s
return s;
}
int main()
{
int a[10],n,i;
scanf("%d", &n);
for(i=0;i<n;i++)
scanf("%d", &a[i]);
printf("%d", _____________); // Function call
return 0;
}
```

SOLUTION

```
#include <stdio.h>
int sum(int a[10], int n)
{
int s=0,i;
for(i=0;i<n;i++ )
s +=a[i];
return s;
}
int main()
{
int a[10],n,i;
scanf("%d", &n);
for(i=0;i<n;i++)
scanf("%d", &a[i]);
printf("%d", sum(a,n));
return 0;
}
```

Point to note:

1. While passing 1-D arrays to functions, we also need to pass “n”.
2. If not, the function will not know how many elements to access in the array.

QUICK EXERCISE

```
#include <stdio.h>
void function(int *p)
{
*p=16;
}
int main()
{
int a[10]={1,2,3,4,5},i;
function(a);
for(i=0;i<5;i++)
printf("%d ", a[i]);
return 0;
}
```

Output:

```
16 2 3 4 5
```

PASSING 2-D ARRAYS TO FUNCTIONS

- Syntax of function definition:
- Way-1: Formal parameters as a pointer

```
void myFunction(int **arr) { . . . }
```

- Way-2: Formal parameters as a sized array

```
void myFunction(int arr[row_size][col_size]) { . . . }
```

- Way-3: Formal parameters as an unsized array

```
void myFunction(int arr[][]) { . . . }
```

- Syntax of function call:

```
myFunction(array_name, rows, cols);
```

QUICK EXERCISE: WRITE A FUNCTION THAT TAKES A STRING PARAMETER AND RETURNS THE NUMBER OF VOWELS IN THE STRING.

```
int vowel (___________) //Formal parameters
{
int count=0,i;
//Calculate the count
return count;
}
int main()
{
char str[10], i;
gets(str);
printf("%d", _____________); // Function call
return 0;
}
```

SOLUTION

```
#include <stdio.h>
int vowel (char *p) //Formal parameters
{
int count=0,i;
for(i=0;p[i];i++)
if(p[i]=='a'||p[i]=='e'||p[i]=='i'|| p[i]=='o'||p[i]=='u')
count++;
return count;
}
int main()
{
char str[10], i;
gets(str);
printf("%d", vowel(str)); // Function call
return 0;
}
```

SUMMARY

- While passing arrays to a function, we pass the array name. Therefore it is “pass by address”.
- Any changes made to the array elements in the function, will be reflected in the array in actual parameter .
- Function call for passing 1-D arrays:

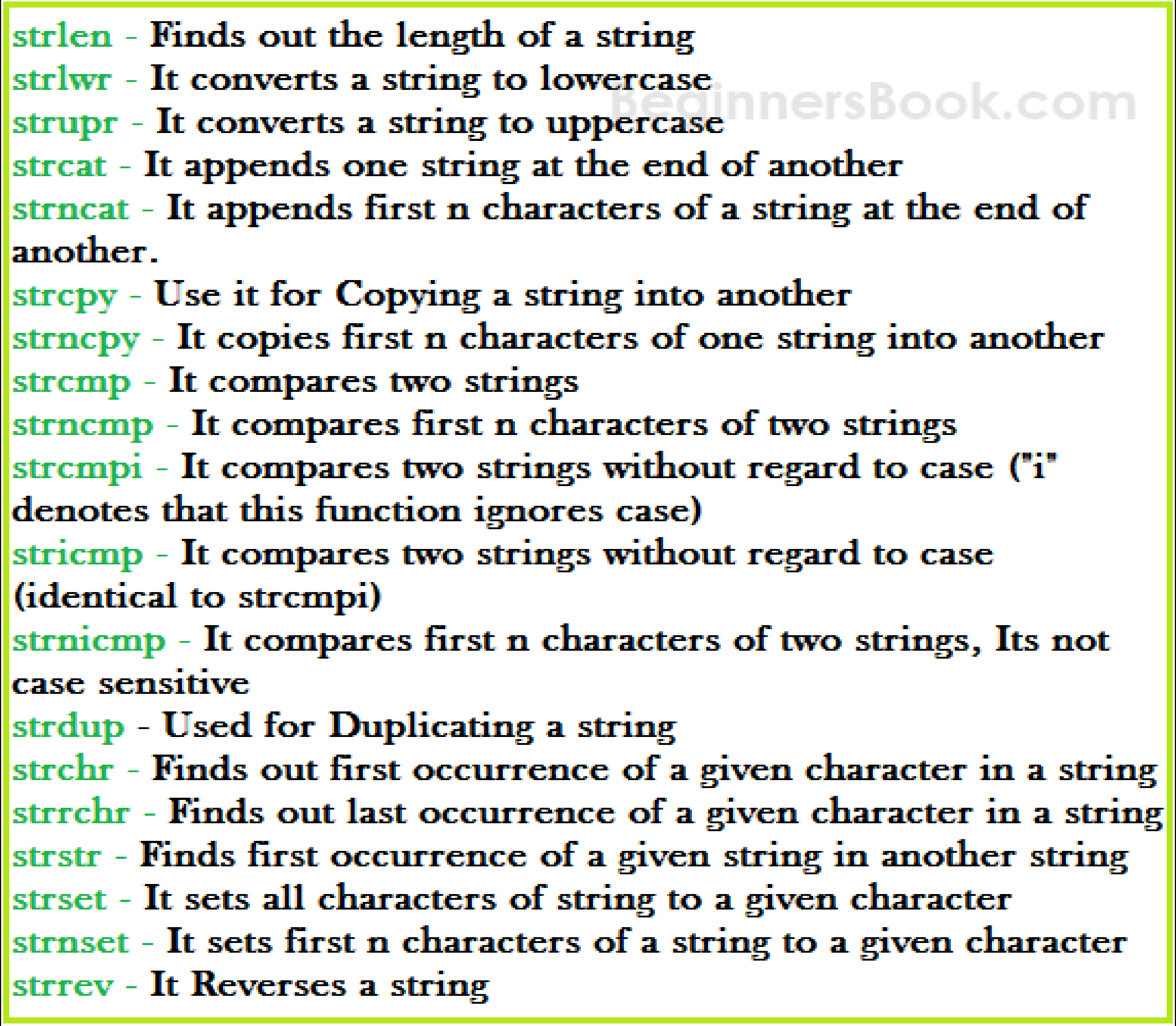
  function_name(array_name, no_of_elements);
- Function call for passing 2-D arrays:

  function_name(array_name, rows, columns);
- Function call for passing strings:

  function_name(string_name);
- While passing strings to functions, the string_name is sufficient.
- No additional information is required because using the starting address all the characters can be accessed until the position of the ‘\0’.

STRING HANDLING FUNCTIONS

- We need to often manipulate strings according to the need of a problem.
- 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".
- Some of the most commonly used string handling operations are copying, concatenation, reversing, calculating length of a string etc.



strlen - Finds out the length of a string
strlwr - It converts a string to lowercase
strupr - It converts a string to uppercase
strcat - It appends one string at the end of another
strncat - It appends first n characters of a string at the end of another.
strcpy - Use it for Copying a string into another
strncpy - It copies first n characters of one string into another
strcmp - It compares two strings
strncmp - It compares first n characters of two strings
strcmpi - It compares two strings without regard to case ("i" denotes that this function ignores case)
stricmp - It compares two strings without regard to case (identical to strcmpi)
strnicmp - It compares first n characters of two strings, Its not case sensitive
strdup - Used for Duplicating a string
strchr - Finds out first occurrence of a given character in a string
strrchr - Finds out last occurrence of a given character in a string
strstr - Finds first occurrence of a given string in another string
strset - It sets all characters of string to a given character
strnset - It sets first n characters of a string to a given character
strrev - It Reverses a string

BeginnersBook.com

strcpy(str1,str2)

- It copies the string str2 into string str1, including the ‘\0’
- Example:

```
#include <stdio.h>
#include <string.h>
int main()
{
char s1[30] = “Hello";
char s2[30] = “World” ;
strcpy(s1,s2); //Note: s2 can also be a string constant
printf("String s1 is: %s", s1);
return 0;
}
```

Output:

```
World
```

strcat(str1,str2)

- It joins s2 at the end of s1.
- Example:

```
#include <stdio.h>
#include <string.h>
int main()
{
char s1[10] = "Hello";
strcat(s1,”World”);
printf("Output string after concatenation: %s", s1);
return 0;
}
```

Output:

```
HelloWorld
```

strcmp(str1,str2)

- This function can return **three different integer values** based on the comparison:
  - **Zero (0):** A value equal to zero when both strings are found to be identical. All of the characters in both strings are same.
  - **Greater than zero (>0):** A value greater than zero is returned when the first non matching character in str1 has a greater ASCII value than the corresponding character in str2 or we can also say str1>str2.
  - **Less than zero (>0):** A value less than zero is returned when the first non matching character in str1 has a lesser ASCII value than the corresponding character in str2 or we can also say str1<str2.

strcmp(str1,str2)

```
#include <stdio.h>
#include <string.h>
int main()
{
char s1[20] = “hello";
char s2[20] = “heLLo";
int i = strcmp(s1,s2);
printf(“%d”, i);
return 0;
}
```

Output:

```
32
```

strcmpi(str1,str2)

- It ignores the case differences.
- Example:

```
#include <stdio.h>
#include <string.h>
int main()
{
char s1[20] = “hello";
char s2[20] = “heLLo";
int i = strcmpi(s1,s2);
printf(“%d”, i);
return 0;
}
```

Output:

```
0
```

strrev (str)

- It reverses a given string.
- Example:

```
int main()
{
char s1[20] = “hello";
strrev(str1);
printf(“%s”, str1);
return 0;
}
```

Output:

```
olleh
```

strlen(str)

- It calculates and returns the length of a string excluding the ‘\0’.
- Example:

```
int main()
{
char s1[20] = “hello";
int len = strlen(str1);
printf(“%d”, len);
return 0;
}
```

Output:

```
5
```

strstr(str1,str2)

- It searches for str2 inside str1.
- If found it returns a pointer pointing to the first position where the string was found.
- Otherwise a null pointer is returned if *str2* is not present in *str1*.
- It is a method to find a substring within a given string
- Example: Finding a word in a large document.

strstr(str1,str2)

```
#include <stdio.h>
#include <string.h>
int main()
{
char s1[40] = "India's population is 1.3 Billion";
char *p = strstr(s1, "populat");
if(p!=NULL)
printf("%s", p);
else
printf("Not found");
return 0;
}
```

Output:

```
population is 1.3 Billion
```

strstr(str1,str2)

```
#include <stdio.h>
#include <string.h>
int main()
{
char s1[40] = "India's population is 1.3 Billion";
char *p = strstr(s1, "1.2");
if(p!=NULL)
printf("%s", p);
else
printf("Not found");
return 0;
}
```

Output:

```
Not found
```

PROGRAM IT

- Create a user-defined function “upper”, that takes a string parameter and converts the string into uppercase.