

# CS - 114 : Computer Workshop

Prof. Chamakuri Nagaiah  
Mahindra-École Centrale, Hyderabad  
[nagaiah.chamakuri@mechyd.ac.in](mailto:nagaiah.chamakuri@mechyd.ac.in)

# The Actual Mechanism

- When an array is passed to a function, the values of the array elements are not passed to the function.
  - The array name is interpreted as the address of the first array element.
  - The formal argument therefore becomes a pointer to the first array element.
  - When an array element is accessed inside the function, the address is calculated using the formula stated before.
  - Changes made inside the function are thus also reflected in the calling program.

# The Actual Mechanism

- Passing parameters in this way is called **call-by-reference**
- Normally parameters are passed in C using **call-by-value**
- Basically what it means?
  - If a function changes the values of array elements, then these changes will be made to the original array that is passed to the function.
  - This does not apply when an individual element is passed on as argument.

# Passing 2-D Arrays

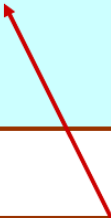
- Similar to that for 1-D arrays.
  - The array contents are not copied into the function.
  - Rather, the address of the first element is passed.
- For calculating the address of an element in a 2-D array, we need:
  - The starting address of the array in memory.
  - Number of bytes per element.
  - Number of columns in the array.
- The above three pieces of information must be known to the function.

## Example : Passing 2-D Arrays

```
#include <stdio.h>

main()
{
    int a[15][25], b[15][25];
    :
    :
    add (a, b, 15, 25);
    :
}
```

```
void add (x, y, rows, cols)
int x[][25], y[][25];
int rows, cols;
{
    :
}
```



We can also write  
`int x[15][25], y[15][25];`

# Return array from function in C

- First point: C programming does not allow to return an entire array as an argument to a function.

# Return array from function in C

- First point: C programming does not allow to return an entire array as an argument to a function.
- However, you can return a pointer to an array by specifying the array's name without an index.
- **Example :** return a single-dimension array from a function

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

# Return array from function in C

- First point: C programming does not allow to return an entire array as an argument to a function.
- However, you can return a pointer to an array by specifying the array's name without an index.
- **Example :** return a single-dimension array from a function

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

- Second point to remember is that C does not advocate to return the address of a local variable to outside of the function, so you would have to define the local variable as static variable.



## Example

```
/* function to generate and return random numbers */
int * getRandom( ) {

    static int    r[10];
    int i;

    /* set the seed */
    srand( (unsigned)time( NULL ) );

    for ( i = 0; i < 10; ++i) {
        r[i] = rand();
        printf( "r[%d]=%d\n", i, r[i]);
    }
    /* return pointer r */
    return r;
}
```

## Example : Contd.

```
#include <stdio.h>

/* main function to call above defined function */
int main () {

    /* a pointer to an int */
    int *p;
    int i;
    /* get a pointer */
    p = getRandom();

    for ( i = 0; i < 10; i++ ) {
        printf( "(p_+_%d)_:_%d\n", i, *(p + i));
    }

    return 0;
}
```

# C Programming Strings

- Array of characters is called a **string**
- Strings are actually one-dimensional array of characters terminated by a null character `\0`.

- Example 1: `char s = "c string tutorial"`

c		s	t	r	i	n	g		t	u	t	o	r	i	a	l	\0
---	--	---	---	---	---	---	---	--	---	---	---	---	---	---	---	---	----

- Example 2: `char s[6] = 'H', 'e', 'l', 'l', 'o', '\0';`

The memory management of the above defined string

Index

0

1

2

3

4

5

Variable

H	e	l	l	o	\0
---	---	---	---	---	----

Address

0x23451	0x23452	0x23453	0x23454	0x23455	0x23456
---------	---------	---------	---------	---------	---------

- Actually, you do not place the null character at the end of a string constant.

# Example

```
#include <stdio.h>

int main () {

    char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
    printf("Greeting message: %s\n", greeting );
    return 0;
}
```

Output» Greeting message: Hello

# Declaration of strings

- Strings are declared in a similar manner as arrays. Only difference is that, strings are of char type.
- Using arrays :

```
char c[] = "abcd";
```

OR,

```
char c[50] = "abcd";
```

OR,

```
char c[] = {'a', 'b', 'c', 'd', '\0'};
```

OR,

```
char c[5] = {'a', 'b', 'c', 'd', '\0'};
```

- Using pointers : `char *p;` OR `char *p = "abcd";`

# Reading Strings

- You can use the **scanf()** function to read a string like any other data types.
- However, the scanf() function only takes the first entered word. The function terminates when it encounters a white space (or just space).
- Example :

```
char c[20];  
scanf("%s", c);
```

## Example

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

### Output

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

# Reading a line of text

- Example: Using `getchar()` to read a line of text

```
#include <stdio.h>

int main(){
    char name[30], ch;
    int i = 0;
    printf("Enter name: ");
    while(ch != '\n')    // terminates if user hit enter
    {
        ch = getchar();
        name[i] = ch;
        i++;
    }
    name[i] = '\0';    // inserting null character at end
    printf("Name: %s", name);
    return 0;
}
```



# Using standard library function

- To make life easier, there are predefined functions **gets()** and **puts()** in C language to read and display string respectively.

```
#include <stdio.h>
int main()
{
    char name[30];
    printf("Enter name: ");
    gets(name);      //Function to read string from user.
    printf("Name: ");
    puts(name);      //Function to display string.
    return 0;
}
```

- **Output** » Enter name: Tom Hanks  
      » Name: Tom Hanks

# Passing Strings to Functions

- Strings are just char arrays

```
#include <stdio.h>

void displayString(char str[]);

int main()
{
    char str[50];
    printf("Enter string: ");
    gets(str);
    displayString(str); // Passing string c to function.
    return 0;
}

void displayString(char str[]){
    printf("String Output: ");
    puts(str);
}
```

# String handling functions

- Just as the character '1' is not the integer 1, the string "123" is not the integer 123. When we have a string of digits, we can convert it to the corresponding integer by calling the standard function `atoi`:

```
char string[] = "123";  
int i = atoi(string);  
int j = atoi("456");
```

- There are various string operations you can perform manually like: finding the length of a string, concatenating (joining) two strings etc.
- But, for programmer's ease, many of these library functions are already defined under the header file **<string.h>**

# Few commonly used string handling functions

## 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
strrev()	Reverses the given string
strlwr()	Converts string to lowercase
strupr()	Converts string to uppercase
strdup()	Duplicates the string
strtok()	Tokenizing/parse given string using delimiter
strstr(str1,str2)	Returns pointer to first occurrence of str2 in str1
strrstr(str1,str2)	Returns pointer to last occurrence of str2 in str1

## Example

```
#include <stdio.h>
#include <string.h>
void main () {
    char str1[12] = "Hello", str2[12] = "World", str3[12];
    int  len ;

    strcpy(str3, str1);/* copy str1 into str3 */
    printf("strcpy( str3, str1) :  %s\n", str3 );

    strcat( str1, str2); /* concatenates str1 and str2 */
    printf("strcat( str1, str2):   %s\n", str1 );

    len = strlen(str1);/* total length of str1 after concatenate */
    printf("strlen(str1) :  %d\n", len );
}
```