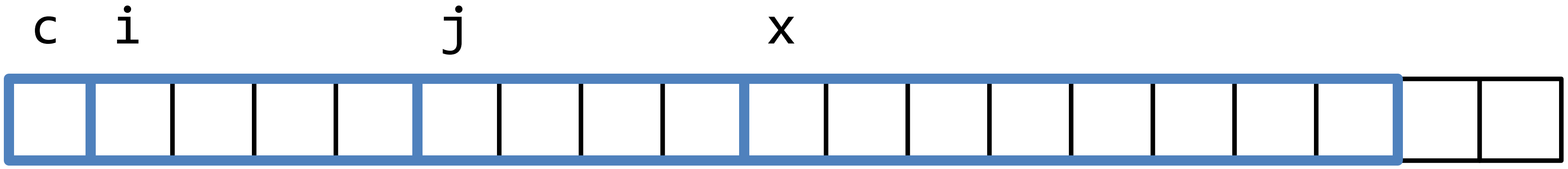
Memory, Arrays & Strings

Memory

```
int main()
{
  char c;
  int i,j;
  double x;
  ….
}
```

We can declare each memory cell separately –

Even with different types



c → char → 1 byte

i, j → int → 4 bytes

X → double → 8 bytes

Arrays

Contents

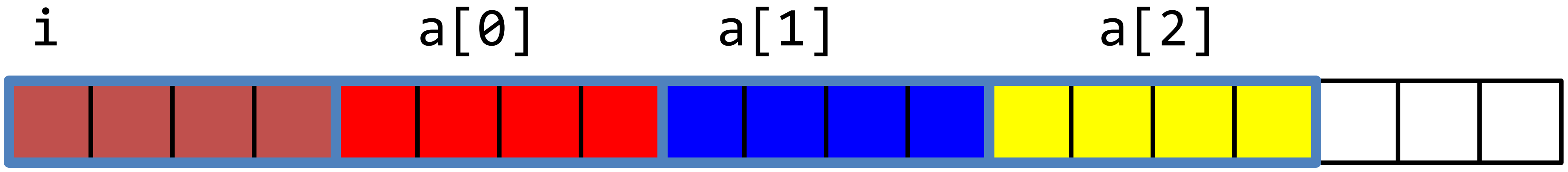
- Arrays
- Multidimensional arrays
- Strings - arrays of characters
- Library functions manipulating strings

Arrays

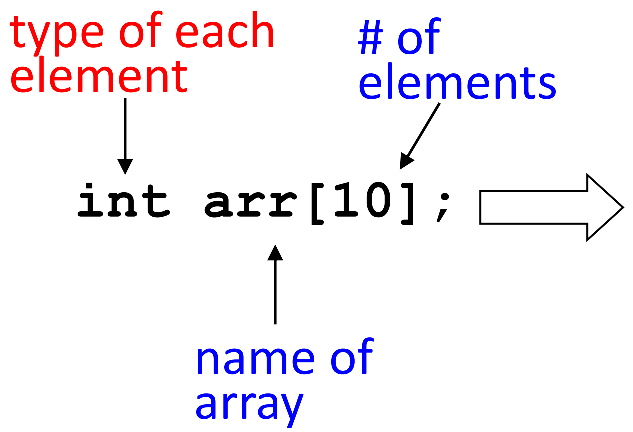
- Defines a block of consecutive cells

-
  ```
  int main()
  {
    int i;
    int a[3];
   .....
   }
  ```

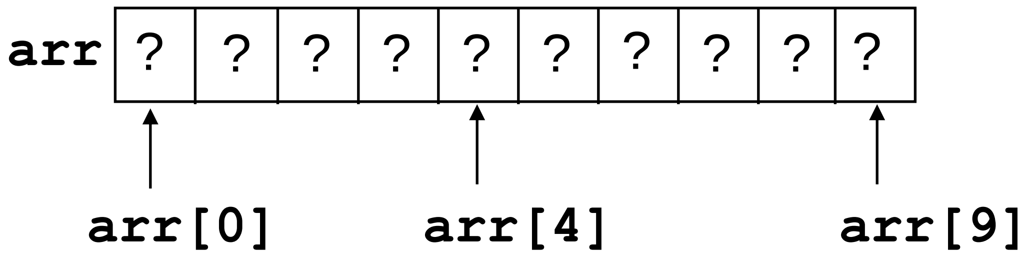
We can declare each cell separately



Array declaration

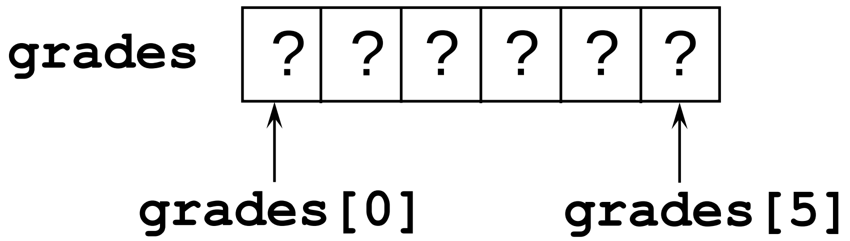


declaration of an array, named **arr**.
consisting of 10 int elements.

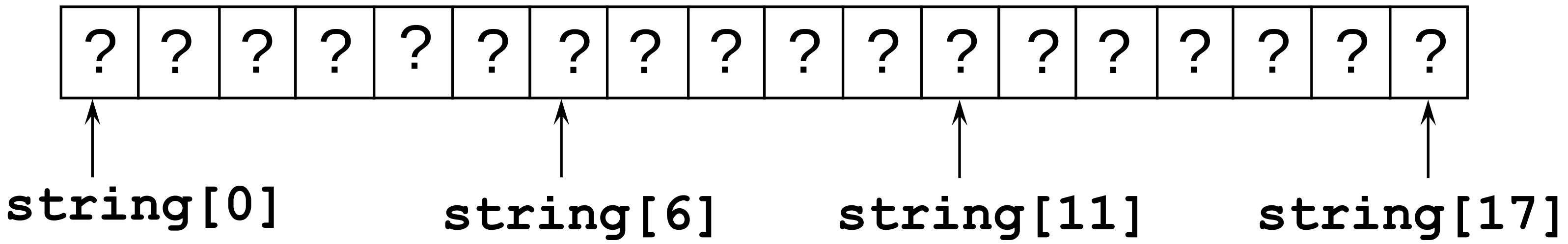


Array Declaration

- *float grades[6];* - declaration of an array, named **grades,** consisting of 6 elements of type float. *sizeof(grades) == 6*sizeof(float).*



- *char string[18];* - declaration of an array, named **string**, consisting of 18 elements of type char. *sizeof(string) == 18*sizeof(char).*



Arrays

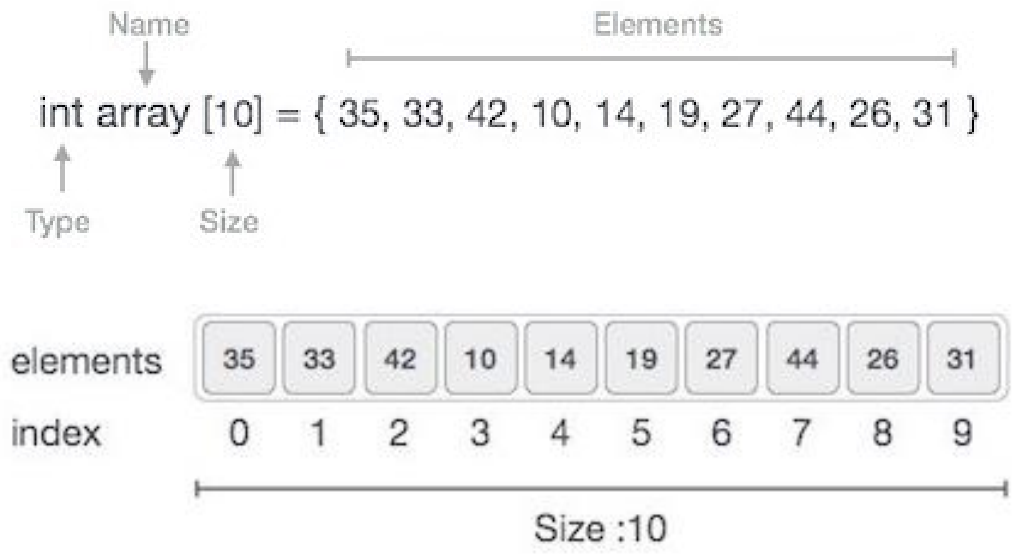
- The 1st position index is 0 (zero).
- C does not have any operator to work with an array “as a whole”.
- Size (number of elements) must be constant in the declaration phase (contrary to running time) – requested amount of memory must be known at the beginning (during compilation time).

```
int a = 5;
int arr[a];      -  wrong, not a constant !
```

- Memory is allocated consecutively.
- No boundary checks are performed by the computer.
- It is the user’s responsibility to perform boundaries check !

  Check if the index is legal.

Arrays



- As per the above illustration, the following important points have to be considered.
  - Index starts with 0.
  - Array length is 10 which means it can store 10 elements.
  - Each element can be accessed via its index.

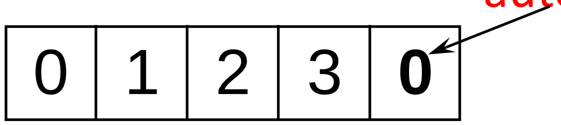
    For example, we can fetch an element at index 6 as 19.

Array Initialization – options

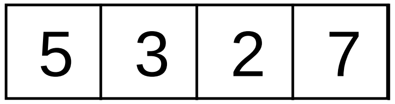
- Looping through the elements and assigning them a value.

```
int arr[5], i;
for (i = 0 ; i < 4 ; ++i)
        arr[i] = i;
```

automatically initialized to zero.



- Within declaration (using an initialization list) - int arr[5] = {5,3,2,7};



- If the size of array is not specified ([]), size is determined by the number of initializing values - int arr[] = {5,3,2,7};

Array Initialization - summary

- // Works

  ```
  int arr[3] = {3, 4, 5};
  ```

- // Works

  ```
  int arr[] = {3, 4, 5};
  ```

- // Init all items to 0

  ```
  int arr[3] = {0};
  ```

- // Bad style - The last one will be automatically 0

  ```
  int arr[4] = {3, 4, 5};
  ```

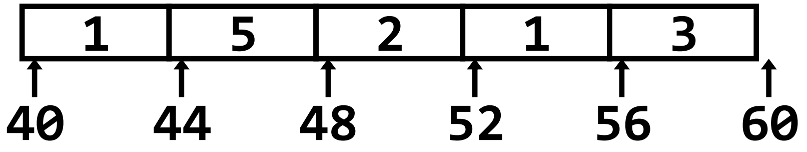
- **// Does not compile**

```
int arr[2] = {3, 4, 5};
```

Arrays

- In the declaration phase –

  int arr[5] = { 1, 5, 2, 1 ,3 }; /*all bytes will be located sequentially but we don't know where, will be automatically assigned by the computer*/



- In the arithmetic operations phase –
  - Arr[n] – n is the exact location in the array
    - arr[0]   40+0*sizeof(int) = 40
    - arr[3]   40+3*sizeof(int) = 52
    - arr[i]   40+i*sizeof(int) = 40 + 4*i
    - arr[-1] 40+(-1)*sizeof (int) = 36
    - a[-1] = 0; // will run but can give unpredictable results – programmer's role to check out of bounds assignments

Example: counter.c

```c
/* counter.c
This program asks the user to enter characters, and counts
them by three groups : digits, white spaces and others.
The digits are counted per digit, which means that the program
will know how many times you entered each digit.
This could be achieved by holding 3 counters, but the program
does it using an array, which is a lot more
elegant and simple. */

#include <stdio.h>

void main(void)
{
        int i, nwhite = 0, nother = 0; /* counters */
        int c; /* for input from the user */
        int digits[10]= {0}; /*initializing the first element
        (or more) automatically initializes the rest to 0*/
        printf("Enter characters and then press CTRL/Z and ENTER\n");
```

Example: counter.c – cont’d

```
        while ( (c=getchar()) != EOF)
                if (c >= '0' && c <= '9')
                /* the character is a digit */
                        digits [c]++;
                /* increase the correct element of arr */
                else
                        if (c==' '||c=='\n'||c=='\t')
                        /*the character is a white space*/
                                nwhite++;
                        else /* the character is not a digit nor a white */
                                nother++;

/* print how many times each digit was entered */
  printf("digits: 0  1  2  3  4  5  6  7  8  9\n     ");
        for (i = 0 ; i < 10 ; i++)
                printf("%d", digits[i]);
        /*print how many whites/other characters were entered */
        printf("\nwhite spaces: %d", nwhite);
        printf("\nothers: %d\n", nother);
}
```

How can we do that without using an array?

Basic array operations

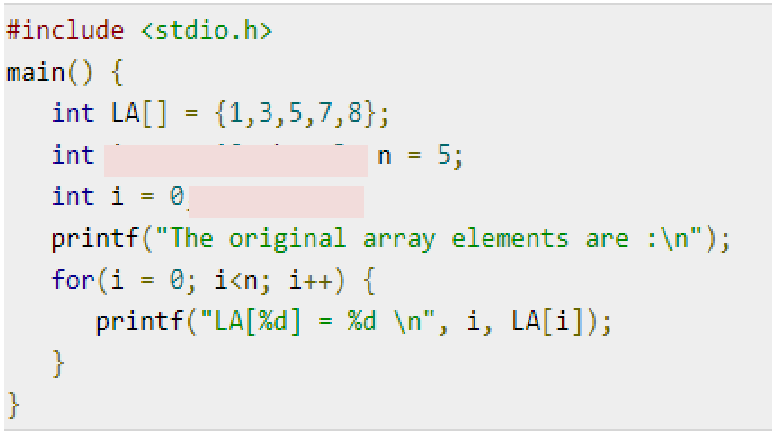
- **Traverse** − print all the array elements one by one.
- **Insertion** − Adds an element at the given index.
- **Deletion** − Deletes an element at the given index.
- **Search** − Searches an element using the given index or by the value.
- **Update** − Updates an element at the given index.

Array Traverse

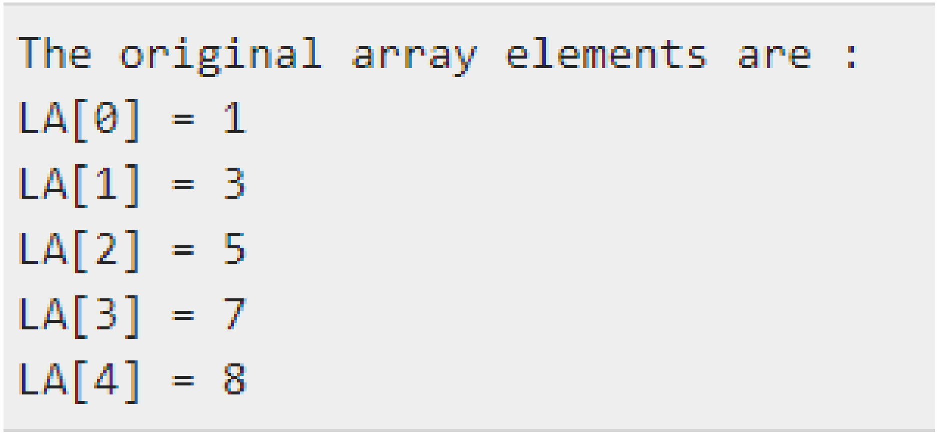
- **Traverse** – print all the array elements one by one.
- Write a program that does traverse to a 5-digits array

Array Traverse - solution

- **Traverse** – print all the array elements one by one.
- Write a program that does traverse to a 5-digits array



```c
#include <stdio.h>
main() {
   int LA[] = {1,3,5,7,8};
   int [redacted] n = 5;
   int i = 0[redacted]
   printf("The original array elements are :\n");
   for(i = 0; i<n; i++) {
      printf("LA[%d] = %d \n", i, LA[i]);
   }
}
```



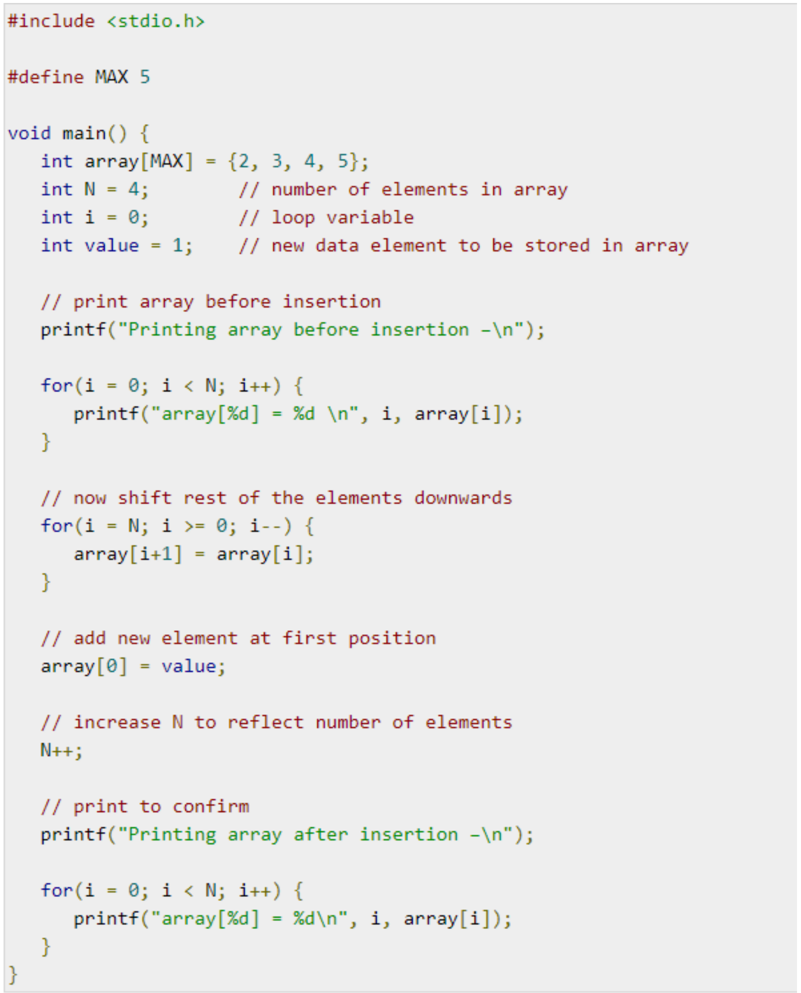
```
The original array elements are :
LA[0] = 1
LA[1] = 3
LA[2] = 5
LA[3] = 7
LA[4] = 8
```

Array Insertion

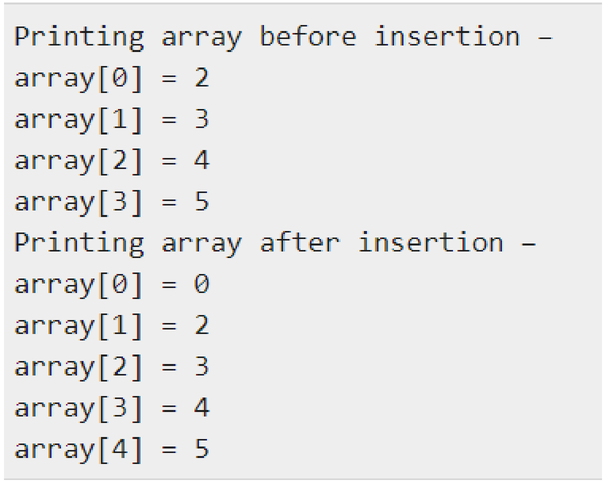
- **Insertion** – adds an element at the given index.
- Write a program that insert the number 0 in the beginning of a 4-digits array

Array Insertion - solution

- **Insertion** – adds an element at the given index.
- Write a program that insert the number 0 in the beginning of a 4-digits array



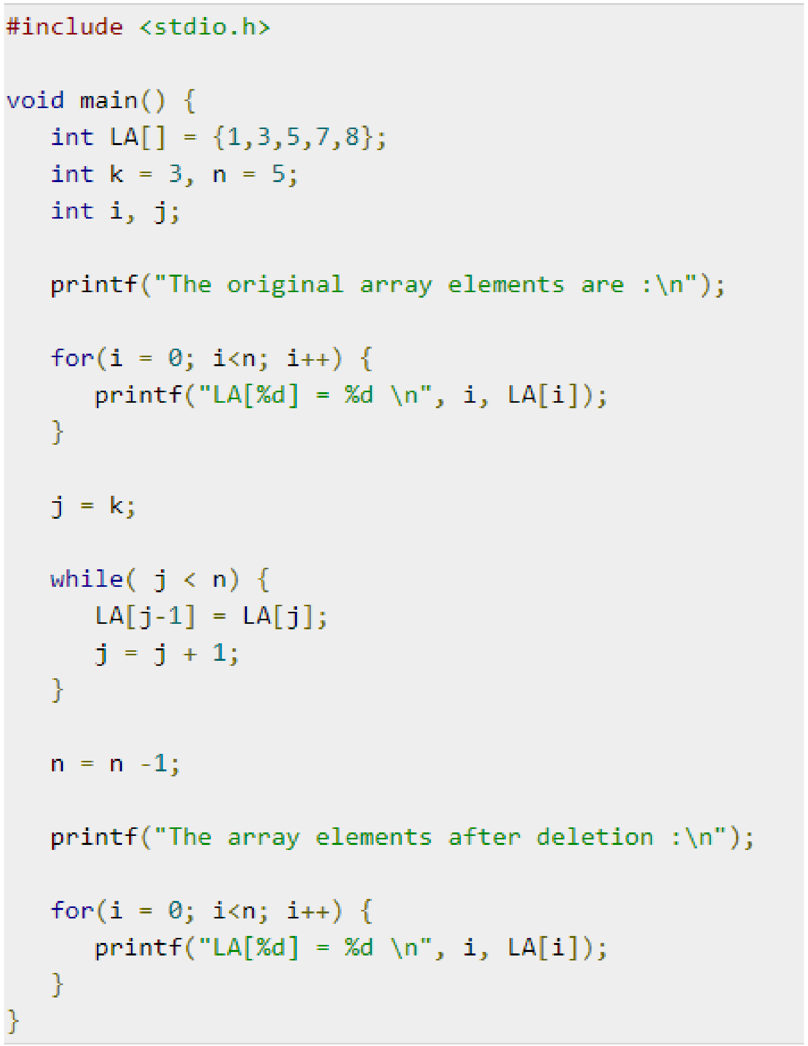
```c
#include <stdio.h>

#define MAX 5

void main() {
   int array[MAX] = {2, 3, 4, 5};
   int N = 4;         // number of elements in array
   int i = 0;         // loop variable
   int value = 1;     // new data element to be stored in array

   // print array before insertion
   printf("Printing array before insertion -\n");

   for(i = 0; i < N; i++) {
      printf("array[%d] = %d \n", i, array[i]);
   }

   // now shift rest of the elements downwards
   for(i = N; i >= 0; i--) {
      array[i+1] = array[i];
   }

   // add new element at first position
   array[0] = value;

   // increase N to reflect number of elements
   N++;

   // print to confirm
   printf("Printing array after insertion -\n");

   for(i = 0; i < N; i++) {
      printf("array[%d] = %d\n", i, array[i]);
   }
}
```



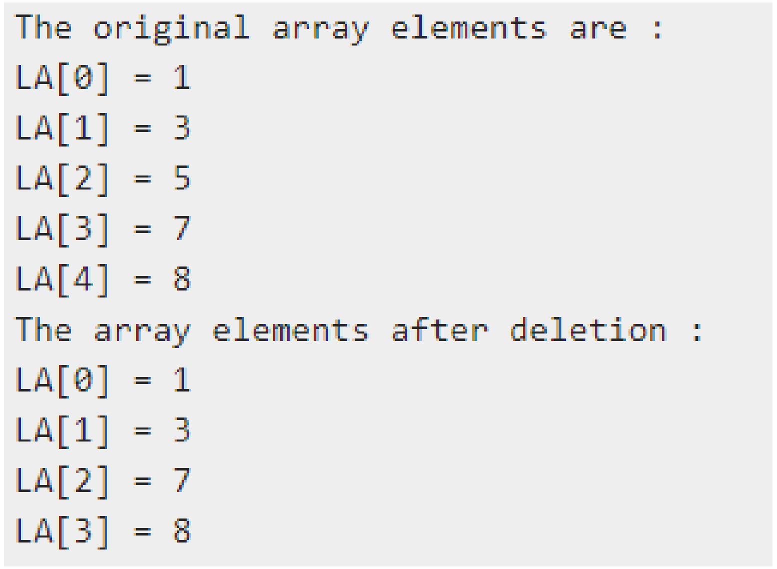
```
Printing array before insertion –
array[0] = 2
array[1] = 3
array[2] = 4
array[3] = 5
Printing array after insertion –
array[0] = 0
array[1] = 2
array[2] = 3
array[3] = 4
array[4] = 5
```

Array Deletion

- **Deletion** – deletes an element at the given index.
- What is this program doing



```c
#include <stdio.h>

void main() {
   int LA[] = {1,3,5,7,8};
   int k = 3, n = 5;
   int i, j;

   printf("The original array elements are :\n");

   for(i = 0; i<n; i++) {
      printf("LA[%d] = %d \n", i, LA[i]);
   }

   j = k;

   while( j < n) {
      LA[j-1] = LA[j];
      j = j + 1;
   }

   n = n -1;

   printf("The array elements after deletion :\n");

   for(i = 0; i<n; i++) {
      printf("LA[%d] = %d \n", i, LA[i]);
   }
}
```



```
The original array elements are :
LA[0] = 1
LA[1] = 3
LA[2] = 5
LA[3] = 7
LA[4] = 8
The array elements after deletion :
LA[0] = 1
LA[1] = 3
LA[2] = 7
LA[3] = 8
```

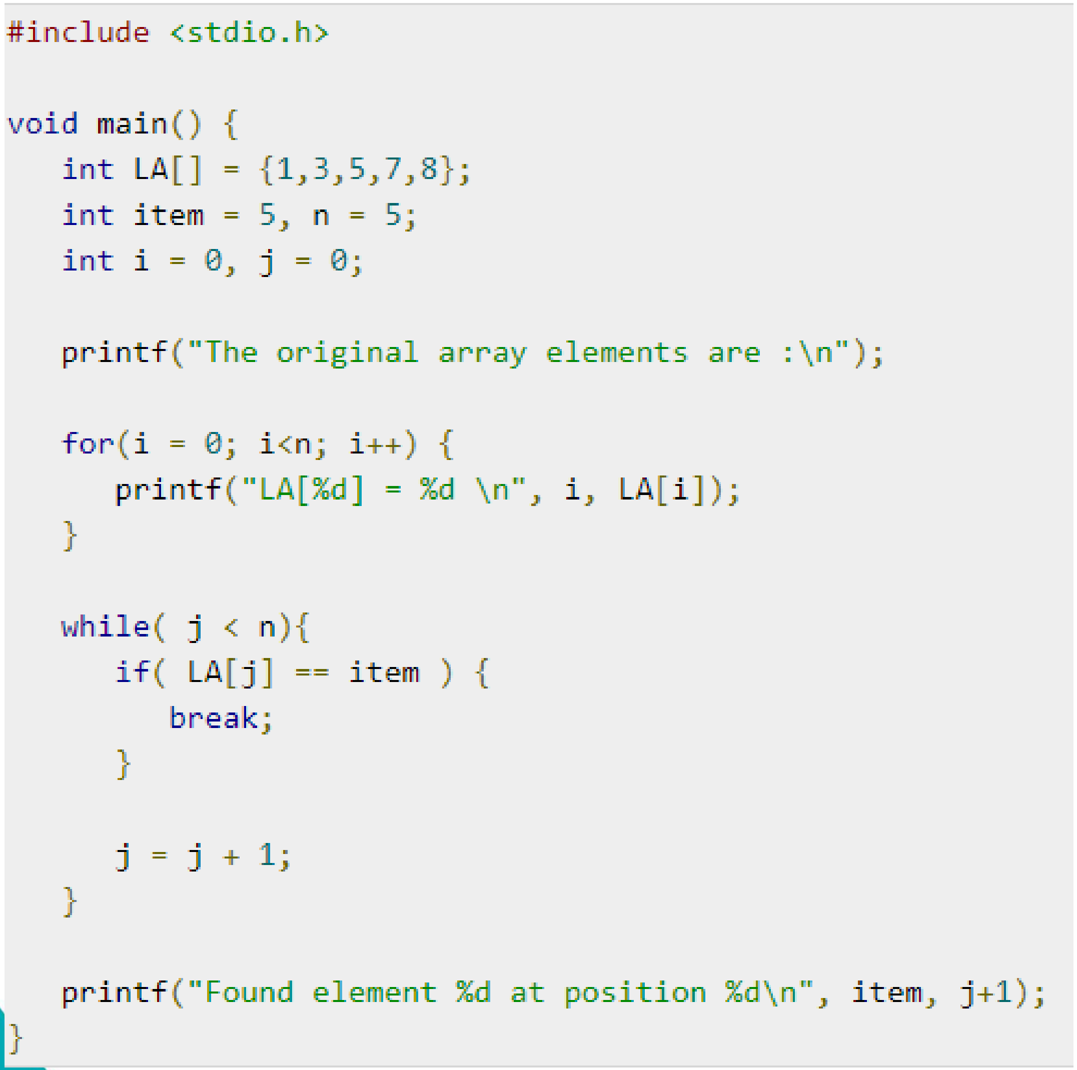
LA2, LA1 or LA[4]=0

Array Search

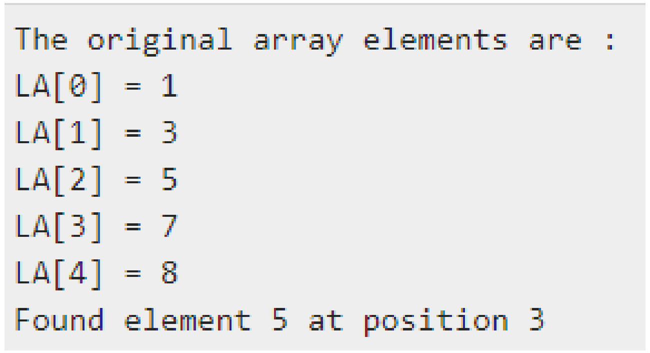
- **Search** − searches an element using the given index or its value
- Write a program that search the number 5 in a {1,3,5,7,8}

Array Search - solution

- **Search** – searches an element using the given index or its value
- Write a program that search the number 5 in a a sorted array {1,3,5,7,8}



```c
#include <stdio.h>

void main() {
   int LA[] = {1,3,5,7,8};
   int item = 5, n = 5;
   int i = 0, j = 0;

   printf("The original array elements are :\n");

   for(i = 0; i<n; i++) {
      printf("LA[%d] = %d \n", i, LA[i]);
   }

   while( j < n){
      if( LA[j] == item ) {
         break;
      }

      j = j + 1;
   }

   printf("Found element %d at position %d\n", item, j+1);
}
```



```
The original array elements are :
LA[0] = 1
LA[1] = 3
LA[2] = 5
LA[3] = 7
LA[4] = 8
Found element 5 at position 3
```

TA exercises (arrays)

- Write a program that:
  - Reads 5 student’s grades (students are numbered 1 to 5). For each student, prompt the user to enter grade as follows:
    - Enter grade for student #1=>
    - Enter grade for student #2 =>   etc.
  - Calculate the average and print with precision of two digits after the decimal point.
  - Print with a suitable header a list of grades above the average.

TA exercises (arrays)

- Write a program that accepts student's grades until -1 is encountered (but a max. of 26 grades because of screen restrictions) and prints out a histogram ('*' represents 10 'points' ; for instance for grade 65, a column of 7 '*'s will be printed).

- Write a program that prints out the prime numbers less than 1000. An array of 1001 elements (element #0 is not used, neither is element #1) is initialized to 1.

TA exercises (arrays)

- Write a program that prints out the 10 first lines of the “Pascal triangle”.

  Hint: two methods for solution:

  1. Use two arrays, one for the current line and one for the next one.

  2. Use one array. It will be easier for you to print:

```
        1
      1 1
    1 2 1
  1 3 3 1
1 4 6 4 1
```

TA exercises (arrays)

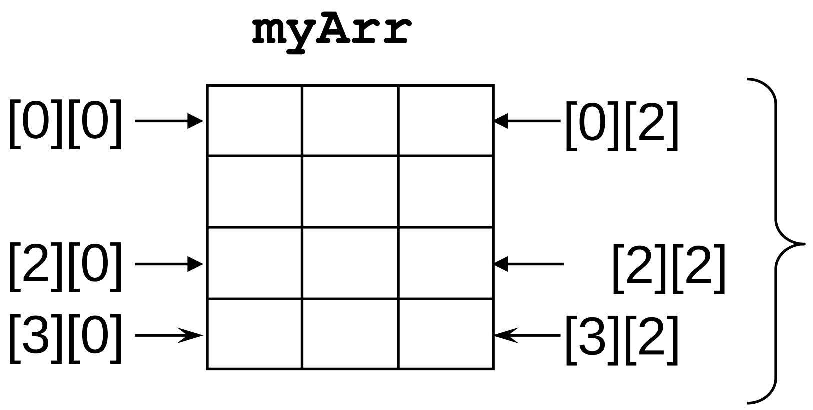
- Write a program that helps draw numbers, in the range 1-99, for a Bingo game. It cannot choose the same number more than once. The algorithm:
  - Initialize the array with all the numbers.
  - Mix the array by changing the contents of two random elements within the array 500 times.
  - Print out the array in order (the numbers are not in order anymore).

Multidimensional Array– Declaration

- Declaration:

```
int  myArr[4][3];

char  TicTacToe[3][3];
```

Declaration of a two-dimensional array named **myArr**, consisting of 12 integer elements



two dimensional representation is for easier viewing only.

Array Initialization – multidimensional

- // Good, works same as arr[2][3]

```
int arr[][3] = {{2,5,7},{4,6,7}};
```

Rows   Columns

- // Bad style, but same as above

```
int arr[2][3] = {2,5,7,4,6,7};
```

- **// Does not compile** → **how can we solve that ?**

```
int arr[3][2] = {{2,5,7},{4,6,7}};
```

arr[2][3] or {2,5,7,4,6,7};

Multidimensional Array - Initialization

- Initialization:

  Looping through the elements and assigning them a value.

```
int myArr[4][3], i, j;
for (i = 0 ; i < 4; ++i)
        for(j = 0 ; j < 3 ; ++j)
                myArr[i][j]= i * j;
```

- Within declaration

```
int myArr [4][3]={1, 2, 3, 4, 5, 6, 7, 8, 10};
```

1	2	3
4	5	6
7	8	9
10	0	0

Example: multiarr.c

- Write a program to check if an input 3*3 matrix is a magic square or not

Solution – magic square

```
/* multiarr.c
This program asks the user to fill in the values for a two dimensional array(3*3),
and checks whether it is a magic square */

#include <stdio.h>
#define TRUE 1
#define FALSE 0

void main()
{
        int mat[3][3]; /* the two dimensional array */
        int i, j;    /* counters and indexes */
        int sum;  /* the sum of each row, column and diagonal */
        int flag = TRUE; /* if the matrix turns out to be an
        ordinary one, and not a magic square, flag will turn FALSE.
        The program will stop checking the square the moment this flag
        becomes FALSE */
        printf ("enter values for the matrix to find out if it is a magic square:\n");
```

Solution – magic square

```
for ( i = 0 ; i < 3 ; i ++)
    {
        for ( j = 0 ; j < 3 ; j++)
        {
            printf("Enter a value for location [%d][%d] => ", i, j);
            scanf ("%d", &mat[i][j]);
        }
    }
    /* check the sum of one diagonal. All sums will be compared to it */
    sum = mat[0][0] + mat[1][1] + mat[2][2];

    /* if the sum of the other diagonal is different, it is not a magic square. In that
    case, put the value FALSE in flag */

    if (sum != mat[0][2] + mat[1][1] + mat[2][0])
        flag = FALSE;
```

Solution – magic square

```
/* check each row and each column of the matrix. If the sum of one of them is different, it is
not magic square*/

        for(i = 0 ; flag == TRUE && i < 3 ; i++)
         {
                    if( mat[i][0] + mat[i][1] + mat[i][2] != sum )
                              flag = FALSE;
                    if( mat[0][i] + mat[1][i] + mat[2][i] != sum )
                              flag = FALSE;
         }

         /* check flag and print a message accordingly */

        if(flag == TRUE)
                    printf("This is a magic square !\n");
         else
                    printf("This is not a magic square !\n");
}
```

TA exercises (Multidimensional Arrays)

- Matrix B - the transpose of matrix A, has the rows of A as columns, and the columns of A as rows.
  Write a program that reads a matrix (3 lines, 4 columns- each entry of type int) and prints it. The program will build the transpose of the matrix and print it. Write the program so that it will be very easy to change to deal with a matrix of any other dimensions (use #define).

- Improve program multiarr. c so that it will be able to check if any matrix of size n*n (n must be odd, of course) is a magic square (the program muliarr.c checks only matrixes of size 3*3).

TA exercises (Multidimensional Arrays)

- Write a program for allocating seats in a movie theater. The hall has 200 seats, arranged in 10 rows * 20 columns. The program should accept data in a loop; it requests for a number of adjacent seats (between 1-20), and prints the locations of chosen seats (or a message if it fails to fulfill the request).

- Write a program that reads the distance between cities in a group of 6 cities. Then, it should accept a trip route between these cities and output the total distance of the trip route.

אוניברסיטת
אריאל
בשומרון

Strings

String Declaration

- C does not specifically define a string type.
- An array of characters (char) followed by a ‘\0’ (ASCII 0) represents a string.
- Strings (contrary to arrays) are always terminated by the null character (‘\0’)
- When you create your own strings:
  - remember it’s there
  - allocate memory for it
  - specify it when you initialize char by char
- All standard library functions comply with strings.
- Declaration:
  - char s[15];

```
char* text = "string";
// means text[5] = g and text[6] = \0
// 7 chars are allocated!
```

String Initialization

- Initialization:

```
char s[15] =  "string example";
char s[] =    "string example";
char s[] = {'s','t','r','i','n','g', ,'e','x','a','m','p','l','e','\0'};
```

'\0' will be automatically inserted but we have to save memory for it

- Generally speaking, "a" is a constant string (containing 2 characters: 'a' and '\0'). however 'a' is a constant character (just 1 character).

- All the initializations above produce the following :

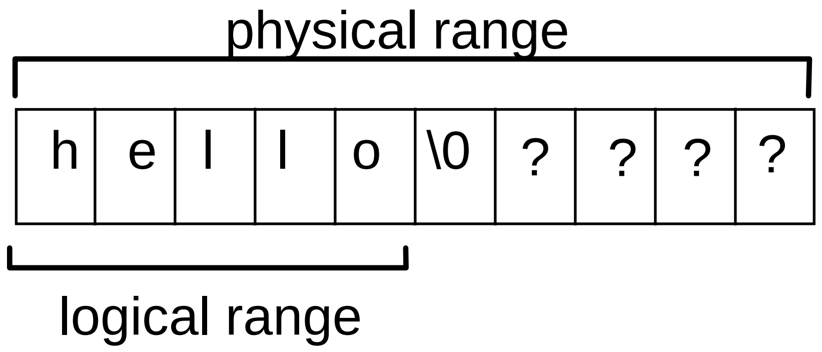
s	t	r	i	n	g		e	x	a	m	p	l	e	\0

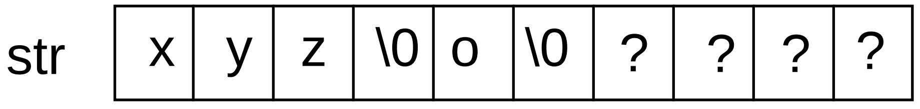
- '\0' is a string terminator (often called "null terminator").

  It is one byte where all bits are zeros.

Physical Range vs. Logical Range

```
char str[10];
strcpy(str, "hello");
```





- strcpy (str, “xyz”);
- strlen(str) is now 3 – the logical size. The physical size remains 10.
- What is the BUG?

No space for the ‘\0’ ending

```
char hello[5];
strcpy(hello, "Hello");
printf("%s",  hello);
```

Strings I/O

- Output:
  - Using stdio.h library functions.
  - One character at a time.

```
char  s[30] = "Hello!";

s may be printed by :

1) printf("%s", s);
2) puts(s);
3) for(i=0 ; s[i]!='\0’ ; ++i)
    putchar(s[i]);
```

- Input:
  - Looping through the elements and assigning them a value.
  - Initialization.
  - Using library functions.

```
S may be give a value by :
1) char  s[30] = "Hello!";
1) scanf("%s" ,&s);
2) gets(s);
```

  - All library functions assume '\0 ' at the end of the string.
  - gets() reads until ‘\n’ (as opposed to scanf()).

Example: str-and-char.c

- Write a program that asks the user to enter strings, and counts how many of these strings begin with the lower case ‘a’ character. These strings are also printed. Strings that begin with other letters are ignored -they are neither printed nor counted.

Solution: str-and-char.c

```
/* str-and-char.c
This program asks the user to enter strings, and counts how many of these
strings begin with the lower case  ‘a’ character. These strings are also printed.
Strings that begin with other letters  are ignored -they are neither printed nor
counted. */

#include <stdio.h>
#define CH  'a'
void main(void)
{
	char str[128];
	int cnt = 0; /* counts the number of strings starting with CH */
	printf("Enter strings (up to 127 characters in a string).\n");
	printf("End a string by pressing ENTER.\n");
	printf("To stop entering strings press CTRL-z  and ENTER.\n\n");

	/* first prompt before the while loop */
	printf("Enter a string.\n");
```

Solution : str-and-char.c – cont’d

אוניברסיטת
אריאל
בשומרון

```
        /* get strings as long as the user doesn’t press  CTRL+z and ENTER. When he
        does, gets() will return NULL(as opposed to getchar())*/

        while (gets(str) != NULL)
        {
                if (str[0] == CH)
        /* if the string starts with ‘a’ */
                {
                        puts(str); /* print the string */
                        cnt++;     /* increase the counter */
                }
                printf("Enter the next string.\n");
        }
        printf("\n\n The number of strings beginning with '%c' is : %d\n",CH, cnt);
}
```

String Manipulating Func

- Input/Output functions - <stdio.h> library
  - All library functions assume '\0' at the end of a string or supply it when required.
  - Functions examples – gets(), puts()
- Analysis of strings' content - <string.h> library
  - strlen(s) - returns the length of string s (not including the null terminator.)
  - strcmp(s1, s2) - compares two strings (lexicographically); Returns:
    - =0 - the strings are equal, >0- first string is greater, <0- second string is greater.
  - strncmp(s1, s2, maxlen) - compares at most maxlen characters of one string with the other(returned value as above).
  - strchr(s, c) - finds the first occurrence of the character c within the string s. Returns the address of this character or NULL if not found.
  - strstr(s1 ,s2) - finds the first occurrence of the string s2 in the string s1. Returns the address where s2 begins in s1, or NULL if not found

String Manipulating Func

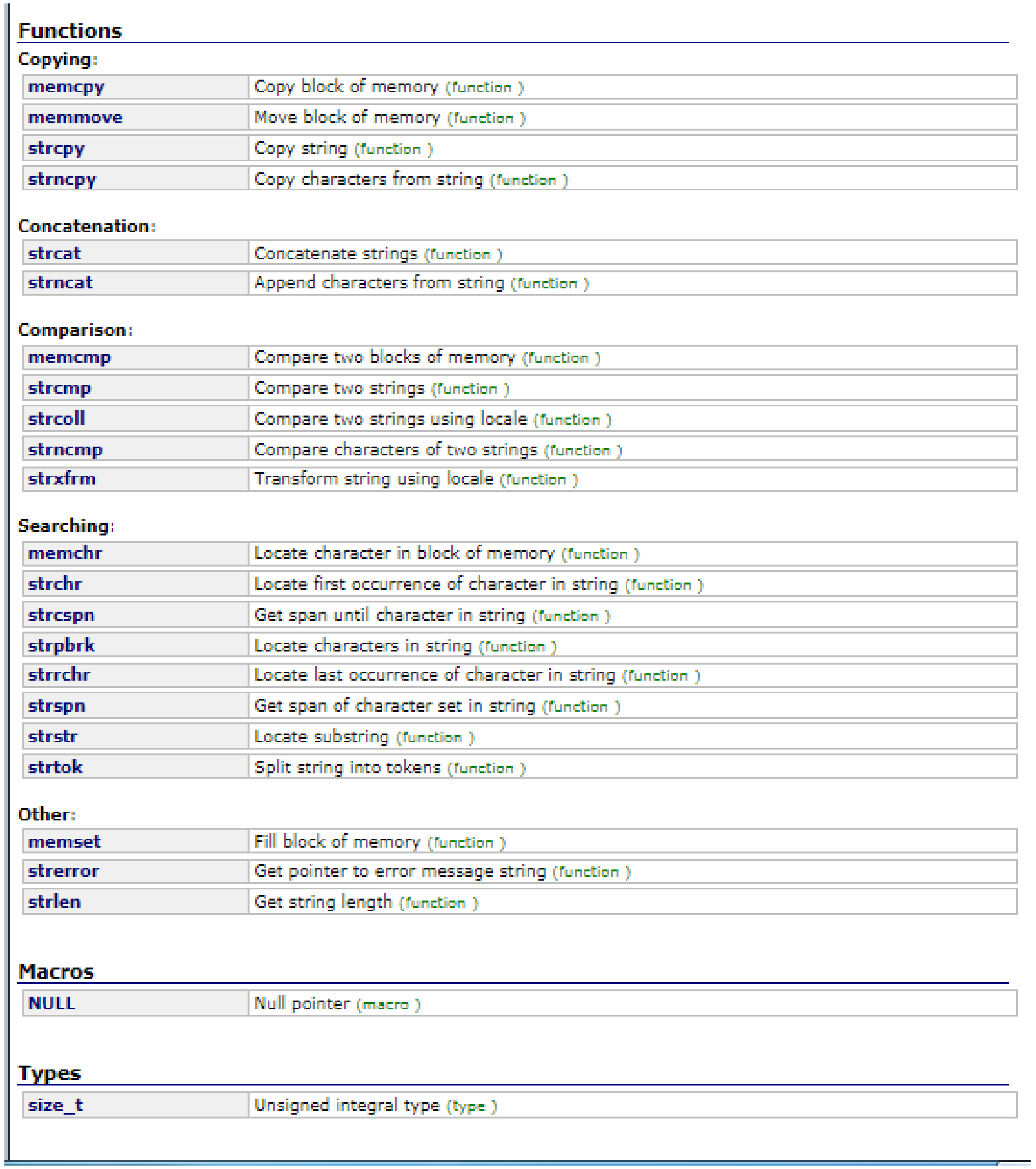
- Analysis of strings’ content (continue)
  - strcpy(s1, s2) - copy s2 into s1. This will start from [0] location
  - strcat(s1, s2) - concatenates s2 to the end of s1. The result is placed in s1.
    - Note:

      No direct assignment is allowed between strings

      Remember, string is not a type in C - it is a convention!
  - strncpy(s1, s2, max) - copy at most max characters from s2 to s1.
  - strncat(s1, s2, max) - appends at most max characters of s2 to s1.

String Manipulating Func

http://www.cplusplus.com/reference/cstring/



Functions

Copying:

memcpy	Copy block of memory (function)
memmove	Move block of memory (function)
strcpy	Copy string (function)
strncpy	Copy characters from string (function)

Concatenation:

strcat	Concatenate strings (function)
strncat	Append characters from string (function)

Comparison:

memcmp	Compare two blocks of memory (function)
strcmp	Compare two strings (function)
strcoll	Compare two strings using locale (function)
strncmp	Compare characters of two strings (function)
strxfrm	Transform string using locale (function)

Searching:

memchr	Locate character in block of memory (function)
strchr	Locate first occurrence of character in string (function)
strcspn	Get span until character in string (function)
strpbrk	Locate characters in string (function)
strrchr	Locate last occurrence of character in string (function)
strspn	Get span of character set in string (function)
strstr	Locate substring (function)
strtok	Split string into tokens (function)

Other:

memset	Fill block of memory (function)
strerror	Get pointer to error message string (function)
strlen	Get string length (function)

Macros

NULL	Null pointer (macro)

Types

size_t	Unsigned integral type (type)

Exercise: inspect-strings.c

- Write a program that illustrates the use of strlen() and strcmp(). The user will enter 2 strings and the program will print their lengths and compares them lexicographically.

Solution: inspect-strings.c

```c
/* inspect-strings.c
This program illustrates the use of strlen() and strcmp(). The user will enter 2 strings and the program will print their lengths and compares them lexicographically (alphabetic). */

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

void main(void)
{
        char str1[48], str2[48];
        int  cmp;

        /* get two strings from the user */
        printf("Enter a string => ");
        gets(str1);
        printf("Enter another string => ");
        gets(str2);
```

Solution: inspect-strings.c – cont’d

```
	/* print the lengths of the strings */
	printf("The length of \"%s\" is : %d\n", str1, strlen(str1));
	printf("The length of \"%s\" is : %d\n\n", str2, strlen(str2));

	/* compare the strings lexicographically */
	cmp = strcmp(str1, str2);
	if ( !cmp)
		printf("The strings are equal.\n");
	else
		if (cmp < 0)
			printf("\"%s\" is greater  than \"%s\"\n", str2, str1);
		else
			printf("\"%s\" is greater than \"%s\"\n", str1, str2);
}
```

Example: mutate-strings.c

```c
/* mutate-strings.c
This program illustrates strcpy(),strncpy()and strcat(). */

#include <stdio.h>
#include <string.h>
void main(void)
{
        char str1[7] = "hey ", /*sizeof(str1) is 7,strlen(str1) is 4*/
        str2[]= "bye bye", /*sizeof(str2) is 8, strlen(str2)is  7 */
        dest[256] = "";  /* sizeof(dest) is 256, strlen(dest) is 0 */
        printf("str1 is \"%s\"\n", str1);
        printf("str2 is \"%s\"\n", str2);
        printf("dest is \"%s\"\n\n", dest);

        strcpy(dest, str1);    /* dest is now "hey " */
        strcat(dest, "world");  /* dest is now "hey world" */
        printf("dest is \"%s\"\n\n", dest);

        strncpy(dest, str2, 3);  /* dest is now "bye world" */
        printf("dest is \"%s\"\n\n", dest);
}
```

Example: del-char.c

- This program receives a string and a char from the user, and deletes all the occurrences of the char from the string. If the char does not occur in the string, a message is printed

del-char.c - solution

```c
/* del-char.c
This program receives a string and a char from the user, and deletes all the
occurrences of the char from the  string. If the char does not occur in the
string, a message is printed */

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

void main(void)
{
        char s[127];  /* the string will be entered by the user */
        int ch;     /* the char to delete from the string s */
        int i, j;      /* counters */
        unsigned int savelen;   /* will hold the length of the original string */
```

del-char.c – solution

```c
    printf("This program gets a string and a character.\n"
           "It deletes all the occurrences of the character from the string.\n");

    /* get the string and the char from the user */
    printf("Enter a string => ");
    gets(s);
    printf("Enter a character => ");
     ch = getchar();

     /*save the length of the string before the deletions*/
     savelen = strlen(s);
     for (i=j=0 ; s[i] ; i++) /*check every char in the string*/
          {
                        if (s[i] != ch)
                        {
                                       s[j] = s[i];
                                       j++;
                        }
          }
```

del-char.c – solution

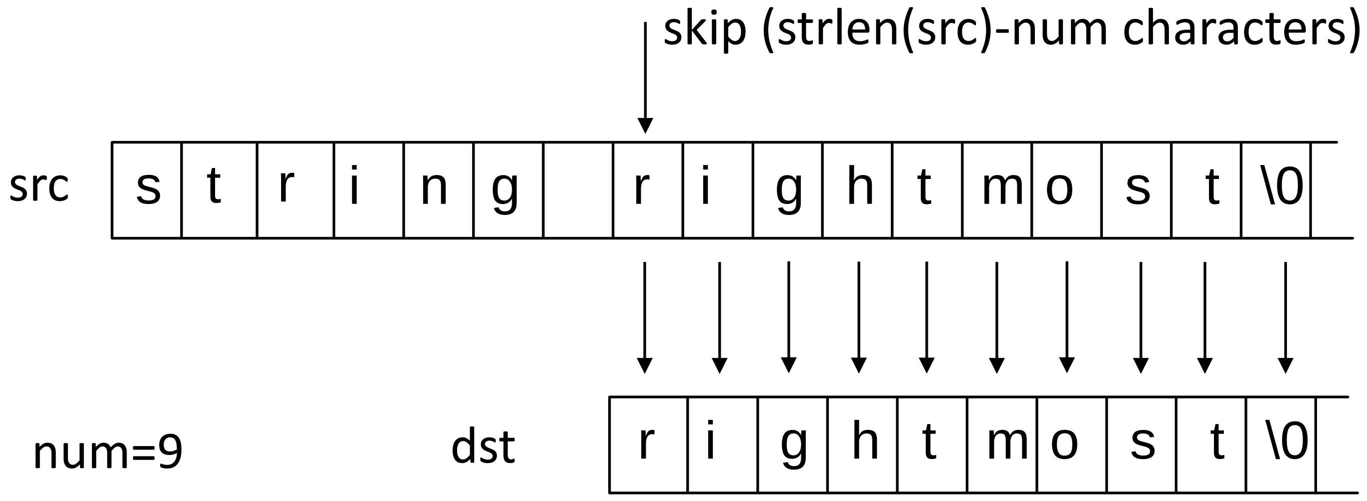
```
    s[j] = '\0';
    /* if ch occurred in s, s is now shorter.  j indicates the new end of the string s, so we
    place a Null terminator there. */

    /* if the length of the original string equals the length of the string after the deletions, it
    means  that the character did not occur in the string */

        if (savelen == strlen(s))
                    printf("The character '%c' does not occur  in the string \"%s\"\n", ch, s);
         else
         {
                    printf("The character '%c' was deleted from the string %d times.\n
                         “The new string is : %s",ch, savelen-strlen(s), s);
         }
}
```

Explanation to right.c

- Explanation:

  Skip number of characters as required. Copy only the rightmost characters

  From that place onwards, copy one character at a time.


skip (strlen(src)-num characters)
src
s t r i n g r i g h t m o s t \0
num=9
dst
r i g h t m o s t \0

- Write a program that implement the right.c and prints a number of the rightmost chars of a string

Solution: right.c

```c
/* right.c
This program prints a number of the rightmost chars of a string */
#include <stdio.h>
#include <string.h>
void main(void)
{
	char src[127], dst[127];
	int num, i, j, length;
	printf("Enter a string: ");
	gets(src);
	printf("How many rightmost chars to copy from the string? =>  ");
	scanf("%d", &num);
	length = strlen(src);
	num = num > length ? length : num;

/* copy num rightmost characters from src to dst */
	for(j = length - num, i=0 ; j <= length ; j++, i++)
			dst[i] = src[j];
	puts(dst);
}
```

Summary – Arrays and Strings

- Array - structure that contains elements of the same type.
- The index of the first element in the array is 0 (zero).
- The size of an array must be fixed. It must be known at compilation time.
- No boundary checks are performed by the computer. Should be done by the user
- No string type in C. However, by convention an array of characters (char) followed by '\0' represents a string.
- Library functions manipulating strings are prototyped in **<string.h>,** except from those dealing with I/O, which are prototyped in **<stdio.h>.**

TA Exercises (strings)

- Write a program that accepts a string and produces another string, which is the original string in reverse order.

- A string that looks the same when read forward and backwards is known as a palindrome. For instance: "ABBA", "madam". Write a program that reads a string and checks whether it is a palindrome. You may use the previous exercise, but it would be very inefficient to do so.

TA Exercises (strings)

- Write a program that accepts a string and an integer, and produces another string that consists of that number of leftmost characters of the original string.

- Write a program that accepts strings (string may be longer than screen line) until NULL (i.e. CTRL/Z) is encountered and prints the longest string entered.

- Write a program that accepts a string and produces a table of the number of occurrences of lower-case letters, UPPER-case letters and digits. Other characters (if any) are not counted by this program.