Passing Arrays to a Function

How to pass arrays to a function?

- An array name can be used as an argument to a function.
  - Permits the entire array to be passed to the function.
  - The way it is passed differs from that for ordinary variables.
- Rules:
  - The array name must appear by itself as argument, without brackets or subscripts.
  - The corresponding formal argument is written in the same manner.
    - Declared by writing the array name with a pair of empty brackets.

An Example with 1-D Array

We can also write

float x[100];

But the way the function is written makes it general; it works with arrays of any size.

```
main()
{
    int  n;
    float   list[100], avg;
    :
    avg  =  average(n,list);
     :
}

float average(a,x)
int a;
float x[];
{
    :
    sum = sum + x[i];
}
```

Same program, with the parameter types specified in the same line as the function definition.

```
main()
{
   int  n;
   float   list[100], avg;
   :
   avg  =  average(n,list);
    :
}

float average(int a, float x[])
{
   :
   sum = sum + x[i];
}
```

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.

Contd.

- 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.

Example: Parameter passed as a value

```
#include <stdio.h>

void swap (int a, int b)
{
  int temp;

  temp=a;
  a=b;
  b=temp;
}
```

```
main()
{
  int x,y;

  x=10;  y=15;
  printf("x=%d y=%d \n",x,y);
  swap(x,y);
  printf("x=%d y=%d \n",x,y);
}
```

Output:

```
    x=10 y=15
    x=10 y=15
```

Example: Minimum of a set of numbers

```
#include <stdio.h>
int minimum (int x[], int y);

main()
{
  int a[100], i, n;

  scanf (”%d”, &n);
  for (i=0; i<n; i++)
    scanf (”%d”, &a[i]);

  printf (“\n Minimum is %d”,
                minimum(a,n));
}
```

```
int minimum (x,size)
int x[], size;
{
  int i, min = 99999;

  for (i=0;i<size;i++)
     if (min < a[i])
        min = a[i];
  return (min);
}
```

Parameter x passed *by reference*, size *by value*.

Example: Square each element of array

```
#include <stdio.h>
void square (int a[], int b);

main()
{
  int a[100], i, n;

  scanf ("%d", &n);
  for (i=0; i<n; i++)
    scanf ("%d", &a[i]);

  square (a, n);

  printf ("\nNew array is: ");
  for (i=0; i<n; i++)
    printf (" %d", a[i]);
}
```

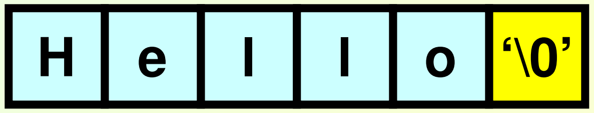
```
void square (x,size)
int x[], size;
{
  int i;

  for (i=0;i<size;i++)
     a[i] = a[i]*a[i];
        min = a[i];

  return;
}
```

Character String

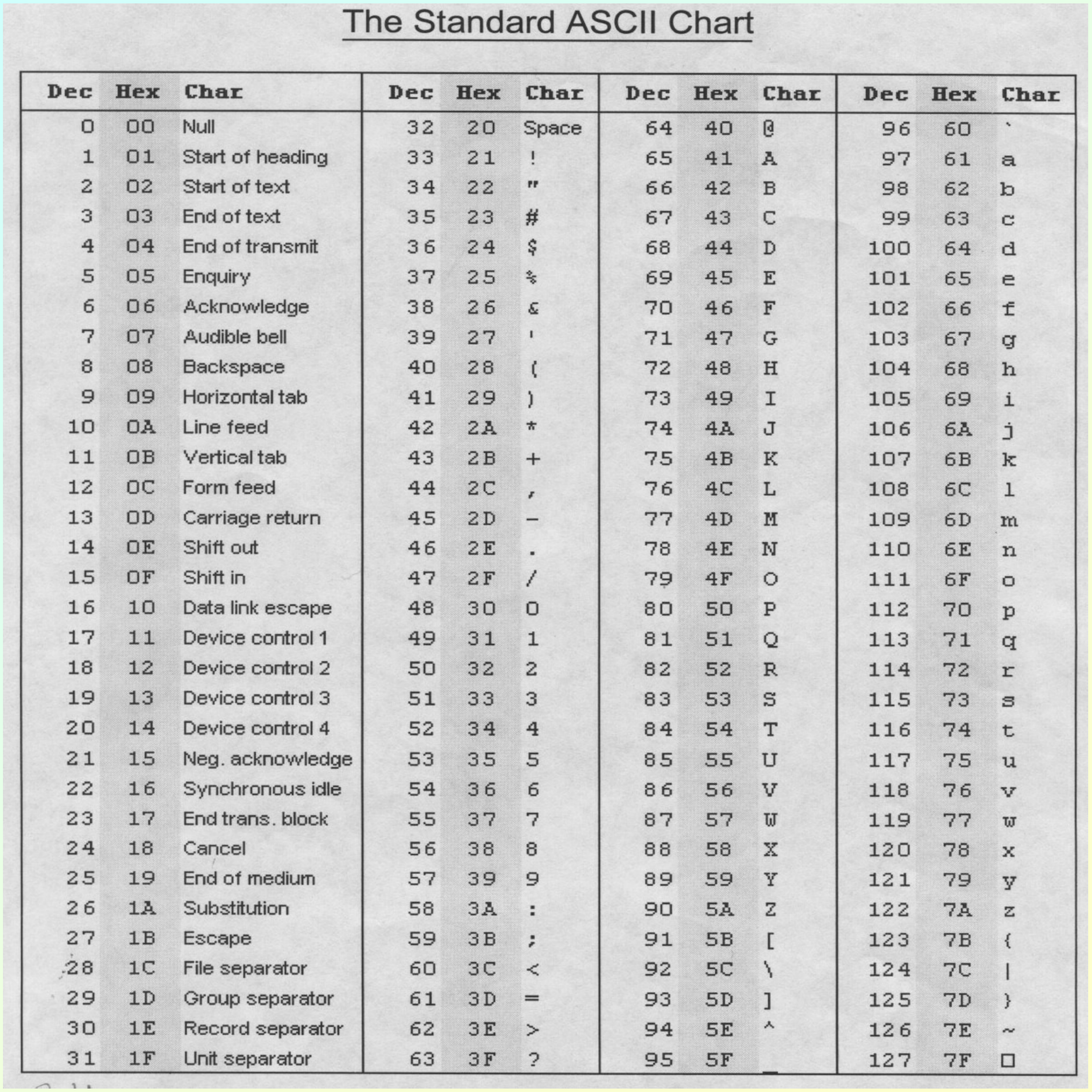
Introduction

- A string is an array of characters.
  - Individual characters are stored in memory in ASCII code.
  - A string is represented as a sequence of characters terminated by the null (‘\0’) character.

“Hello” ➔



ASCII Code Chart



The Standard ASCII Chart

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Declaring String Variables

- A string is declared like any other array:

```
char string-name [size];
```

  - size determines the number of characters in string_name.
- When a character string is assigned to a character array, it automatically appends the null character (‘\0’) at the end of the string.
  - size should be equal to the number of characters in the string plus one.

Examples

```
char   name[30];
char   city[15];
char   dob[11];
```

- A string may be initialized at the time of declaration.

Equivalent (?)

```
char   city[15] = ″Calcutta″;
char   city[15] = {′C′, ′a′, ′l′, ′c′, ′u′,
                          ′t′, ′t′, ′a′};
char   dob[] = ″12-10-1975″;
```

- Hot to access individual characters of a string?
  - Just like a normal array.
  - `city[0], city[1], city[2],` etc.
- Accessing individual characters from a string constant.
  - Possible to do in C.
  - <u>Example</u>: `"GOOD MORNING"[3] will give the value 'D'.`

Reading Strings from the Keyboard

- Two different cases will be considered:
  - Reading words
  - Reading an entire line

Reading "words"

- **scanf can be used with the "%s" format specification.**

```
char    name[30];
 :
 :
scanf ("%s", name);
```

  - **The ampersand (&) is not required before the variable name with "%s".**
    - `name` **represents an address.**
  - **The problem here is that the string is taken to be up to the first white space (blank, tab, carriage return, etc.)**
    - **If we type "Rupak Biswas"**
    - `name` **will be assigned the string "Rupak"**

Reading a “line of text”

- In many applications, we need to read in an entire line of text (including blank spaces).
- We can use the **getchar()** function for the purpose.

```
char   line[81], ch;
int  c=0;
:
:
do
   {
       ch = getchar();
       line[c] = ch;
       c++;
   }
while  (ch != '\n');

c = c - 1;
line[c] = '\0';
```

Read characters until CR ('\n') is encountered

Make it a valid string

Reading a line :: Alternate Approach

```
char line[81];
:
:
scanf ("%[ ABCDEFGHIJKLMNOPQRSTUVWXYZ]", line);
```

➔ Reads a string containing uppercase characters and blank spaces

```
char line[81];
:
:
scanf ("%[^\n]", line);
```

➔ Reads a string containing any characters

Writing Strings to the Screen

- We can use **printf** with the “%s” format specification.

```
char name[50];
:
:
printf ("\n %s", name);
```

Processing Character Strings

- There exists a set of C library functions for character string manipulation.
  - strcpy :: string copy
  - strlen :: string length
  - strcmp :: string comparison
  - strtcat :: string concatenation
- It is required to add the line

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

strcpy()

- Works very much like a string assignment operator.

```
    strcpy (string1, string2);
```

  – Assigns the contents of string2 to string1.

- Examples:

```
    strcpy (city, ″Calcutta″);
    strcpy (city, mycity);
```

- Warning:
  – Assignment operator do not work for strings.

```
        city  =  ″Calcutta″;
```
➔ INVALID

strlen()

- Counts and returns the number of characters in a string.

```
len = strlen (string);
                    /* Returns an integer */
```

  - The null character ('\0') at the end is not counted.
  - Counting ends at the first null character.

```
char  city[15];
int  n;
:
:
strcpy (city, "Calcutta");
n = strlen (city);
```

n is assigned 8

strcmp()

- Compares two character strings.

```
    int strcmp(string1, string2);
```

  – Compares the two strings and returns 0 if they are identical; non-zero otherwise.

- Examples:

```
  if (strcmp(city, ”Delhi”) == 0)
      {   ……   }

  if (strcmp(city1, city2) != 0)
       { …… }
```

strcat()

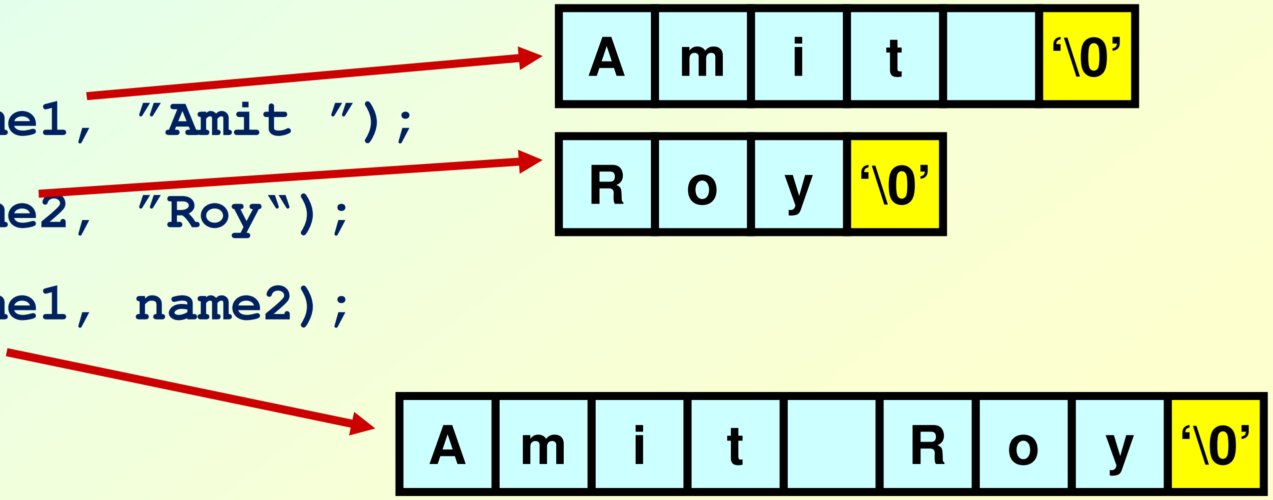
- Joins or concatenates two strings together.

```
strcat (string1, string2);
```

  - string2 is appended to the end of string1.
  - The null character at the end of string1 is removed, and string2 is joined at that point.

- Example:

```
strcpy(name1, ″Amit ″);

strcpy(name2, ″Roy“);

strcat(name1, name2);
```



Example:: count uppercase

```
/* Read a line of text and count the number of
uppercase letters */
#include  <stdio.h>
#include  <string.h>
main()
{
    char  line[81];
    int  i, n, count=0;
    scanf ("%[^\n]", line);
    n = strlen (line);
    for (i=0; i<n; i++)
       if (isupper(line[i])
              count++;
    printf ("\n The number of uppercase letters in
the string %s is %d", line, count);
}
```

Example:: compare two strings

```
#include <stdio.h>

int my_strcmp(char s1[ ],char s2[ ])
{
      int i=0;
        while(s1[i]!='\0' && s2[i]!='\0'){
             if(s1[i]!=s2[i]) return(s1[i]-s2[i]);
             else i++;
        }
      return(s1[i]-s2[i]);
}
```

Parameters passed as character array

Compare character pairs till the end of a string

Return immediately if they are not equal.

```
main()
{
  char string1[100],string2[100];

  printf("Give two strings \n");
  scanf("%s%s",string1,string2);

  printf("Comparison result: %d \n",
           my_strcmp(string1,string2));
}
```

```
Give two strings
IITKGP IITMUMBAI
Comparison result: -2
```

```
Give two strings
KOLKATA KOLKATA
Comparison result: 0
```

Introduction to Pointers

- What is the concept?
  - Pointer is a variable which stores the address in memory location of another variable.
  - When declared, we must specify the data type of the variable being pointed to.
  - Examples:

```
int *p;
float  *x, *y;
char  *flag;
```

- A pointer variable can be assigned the address of another variable.

```
int  a, *p;
a=10;
p = &a;  /* Address of ‘a’ assigned to ‘p’ */
printf (”%d %d”, a, *p);
         /* Will print “10 10” */
```

- Point to note:
  - Array name indicates pointer to first array element.

```
int  num[10], *xyz;
xyz = num;  /* Points to x[0] */
```

- When an integer expression E is added to or subtracted from a pointer, actually scale factor times E is added/subtracted.
  - Scale factor indicates size of the data item being pointed to in number of bytes.
  - Scale factor for char is 1, int is 4, float is 4, double is 8, etc.

```
int a, *p;
p = &a;  /* p is assigned address of a
                                (say, 2500) */
p++;         /* p will become 2504 */
p = p - 10;  /* p will become 2464 */
```

- Consider the declaration:

```
int x[5] = {1, 2, 3, 4, 5};

int *p;
```

  – Suppose that the base address of x is 2500, and each integer requires 4 bytes.

Element	Value	Address
x[0]	1	2500
x[1]	2	2504
x[2]	3	2508
x[3]	4	2512
x[4]	5	2516

Contd.

Both `x` and `&x[0]` have the value `2500`.

`p = x;` and `p = &x[0];` are equivalent.

- Relationship between p and x:

```
p   =  &x[0]  =  2500
p+1 =  &x[1]  =  2504
p+2 =  &x[2]  =  2508
p+3 =  &x[3]  =  2512
p+4 =  &x[4]  =  2516
```

*(p+i) gives the value of x[i]

- **An example:**

```
int  x[ ] = {1,2,3,4,5,6,7,8,9,10};
int  *p;

p = x + 3;  /* Point to fourth element of x */

printf ("%d", *p);  /* Will print 4 */

printf ("%d", *(p+5));
                    /* Will print 9 */

printf ("%d %d", p[3], p[-1]);
                    /* Will print 7 and 3 */
```

Example: function to find average

```
#include <stdio.h>
main()
{
  int x[100], k, n;

  scanf ("%d", &n);

  for (k=0; k<n; k++)
     scanf ("%d", &x[k]);

  printf  ("\nAverage is %f",
              avg (x, n));
}
```

```
float avg (array, size)
int array[], size;
{
  int  *p, i , sum = 0;

  p = array;

  for (i=0; i<size; i++)
      sum = sum + *(p+i);

  return ((float) sum / size);
}
```