20CS1101 –PROGRAMMING FOR PROBLEM SOLVING

(Common to all branches)

UNIT – IV

ARRAYS: Definitions, Initialization, Characteristics of an array, Array Categories.

STRINGS: Declaration and Initialization of strings, String handling functions.

STORAGE CLASSES: Automatic, External, Static and Register Variables.

→Arrays

Introduction:

Consider the following example

```
main()
{
int a=2;
    a=4;
printf("%d" ,a);
getch();
}
```

Output: 4.

In the above example the value of „a" printed is 4. 2 is assigned to „a" before assigning 4 to it. When we assign 4 to „a" then the value stored in „a" is replaced with the new value. Hence ordinary variables are capable of storing one value at a time. But the array variables are able to store more than one value at a time. **Definition of Array**

An array is defined as a set of homogeneous data items of the same type that share a common name.

Element:

The individual values in the array are called as elements.

Index or Subscript:

Each array element is referred by specifying the array name, followed by a number with in square braces referred as an index or subscript.

Note:

Arrays are static data structures.

Declaration of array

The declaration of array is as follows.

Syntax:

Storage type Data type array name [size];

Where storage type may be either auto or register or static or extern. The storage type while declaring an array is optional. The data type specifies the type of element that will be stored in the array. The size is a +ve integer constant. Indicating the maximum number of elements that can be stored in the array.

Example:

```
int days[31];
char book[50];
float real[10];
```

In an n element array the array elements are stored in x[0],x[1]……x[n-2],x[n-1] as shown in fig.

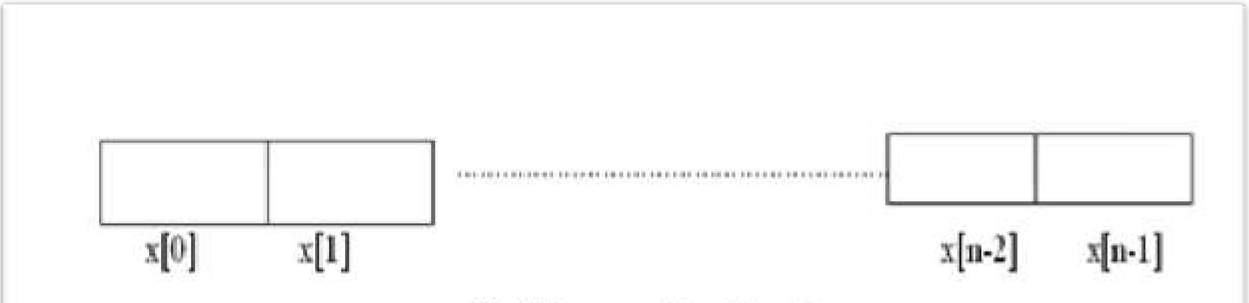


Fig: Memory allocation of arrays.

→Initialization of Arrays

Syntax:

Storage type Data type array name[size]= { list of values };

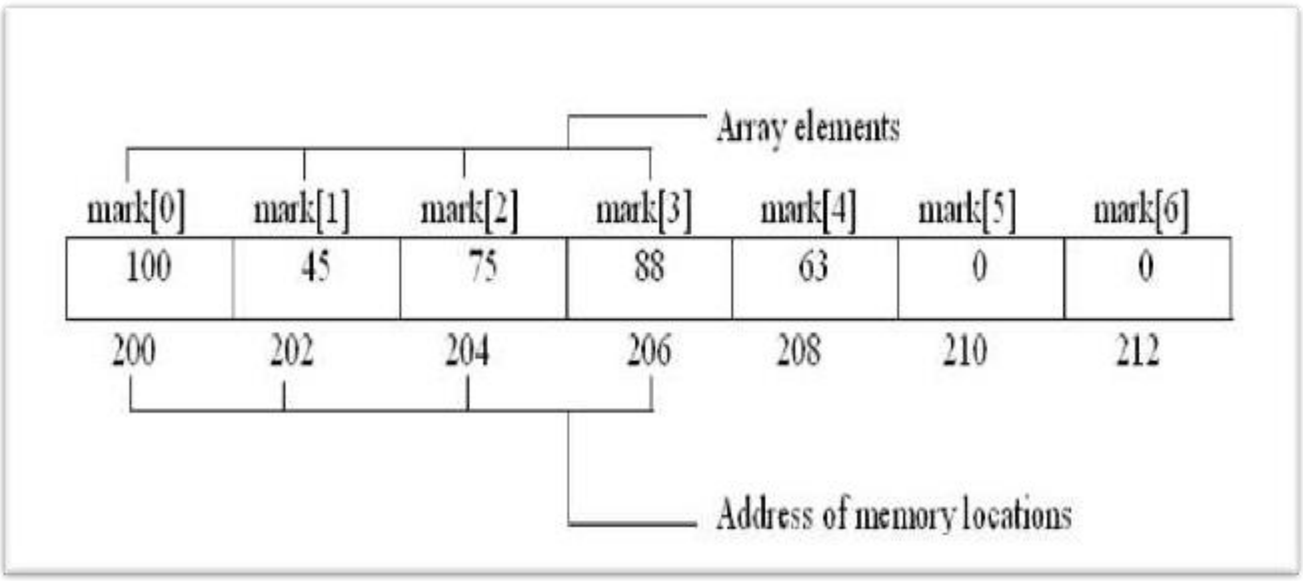
The lists of values are separated by commas used for initializing the array.

Example:

```
int mark [7] = {100,45,75,88,63 };
char name[8] = {„s‟,‟u‟,‟d‟,‟e‟,‟r‟,‟10‟ };
float amount [ ] = { 12.3, 234.56, 34.56, 5768.90 };
```

In the third example the size of the array is omitted. In such cases, the compiler allocates enough memory space for all the elements given for initialization. Hence the size of the array amount [] will be 4.

Fig: Shows the memory occupied by the array mark in the first example.



→Characteristics of Arrays

1 The array name should be a valid identifier.

2 The name of the array should be unique, similar to other variables.

3 The values of the elements stored in the array should be of the same type.

Applications of Arrays

1 Insertion of a value in an existing array at a particular position.

2 Deletion of value from an array.

3 Traversal of an array.

4 Sorting an array in to some particular order.

5 Searching for a value in an array.

6 Merging as two arrays.

→Types of Arrays (Categories of Arrays)

There are 3 types of arrays.

One dimensional arrays

Two dimensional arrays

Multi-dimensional arrays.

1. One dimensional arrays

Arrays whose elements are specified by a single subscript are called as one dimensional arrays or single dimensional or single subscripted or linear arrays.

Syntax:

```
Storage type    Data type     array name [size];
```

Accessing array elements:

Once an array I s declared then the individual elements in the array are accessed with the help of subscripts. All the array elements are numbered starting from zero. The first item is stored at the address pointed by the array name itself.

Example: Write a C program to find the biggest of 10 numbers using arrays.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,n,a[10],max;
clrscr();
printf("enter n value");
scanf("%d",&n);
for(i=0;i<n;i++)
scanf("%d",&a[i]);
max=a[0];
for(i=0;i<n;i++)
if(a[i]>max)
max=a[i];
printf("maximum number=%d",max);
getch();
}
```

2. Double dimensional arrays

A double dimensional array is defined in the same manner as a single dimensional array except that it requires two pairs of square brackets for two subscripts.

Syntax:

Storage type   Data type   array name [row size] [column size];

Example: int mark [2] [2];

Defines a table as an integer array having 2 rows and 2 columns. An array element starts with an index zero and so the individual elements of an array will be

int mark	**[0][0]**	**[0][1]**
	[1][0]	**[1][1]**

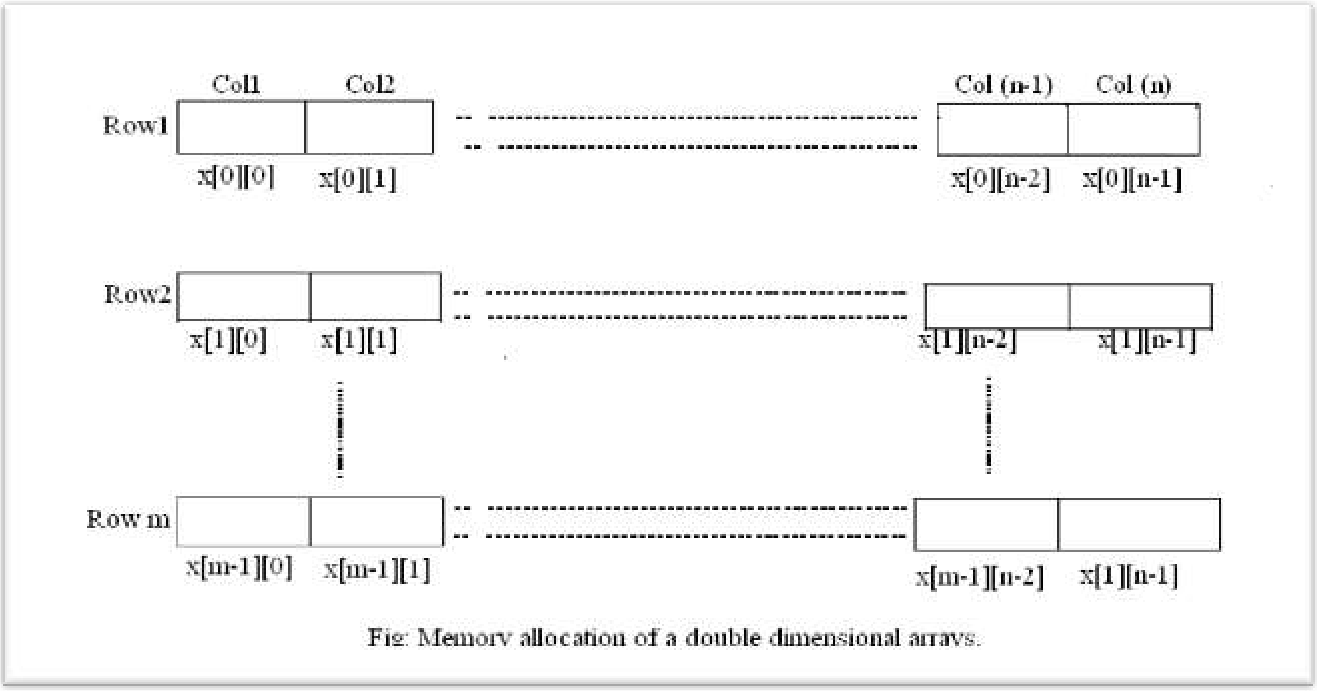


Fig: Memory allocation of a double dimensional arrays.

Initializing double dimensional arrays

Syntax:

Storage type data type array name [row size][columns size]={list of values};

Example:

```
        int    mark[4][2] ={
                        {  84, 56   },
                        {  92, 67   },
                        {  75, 78   },
                        {  69,  89  }
                     };
```

It can be also declared as

int mark[4][2]= { 84, 56, 92, 67, 75, 78, 69, 89 }; The result of the initial assignment are

```
Mark[0][0]=84          mark[1][0]=56
Mark[0][1]=92          mark[1][1]=67
Mark[0][2]=75          mark[1][2]=78
Mark[0][3]=69          mark[1][3]=89
```

Example: Write a C program to calculate the sum of all elements in a matrix using double dimensional array.

```
#include<stdio.h>
#include<conio.h>
void main()
{
```

```
int a[10][10], i,j,m,n,sum=0;
printf("Enter the order of matrix");
scanf("%d%d",&m,&n);
printf("Enter the elements of a matrix");
for(i=0;i<m;i++)
 {
   for(j=0;j<n;j++)
    {
      scanf("%d",&a[i][j]);
     sum=sum+a[i][j];
     }
 }
 printf("Sum of the matrix are %d",sum);
getch();
}
```

3. Three dimensional or multi-dimensional array

The C program allows array of two or multi dimensions.

Syntax: Data type array name [s1][s2][s3] .... [sn]

Initialization:

```
int    mat[3][3][3]=  {
                                {   1, 2, 3,
                                   4, 5, 6,
                                   7, 8, 9,
                                    1, 4, 7,
                                    2, 8, 9,
                                    1, 2, 3
                               }
                        };
                               {
                                        2, 9, 8,
                                        4, 1, 3,
                                        3, 2, 3
                               }
```

A three dimensional array can be thought of as an array of arrays. The outer array contains three elements the inner array size is two dimensional with size [3] [3]

Example: Write to C program to explain the working of three dimensional arrays

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a3d[3][3][3];
int a,b,c;
clrscr();
for(a=0;a<3;a++
for(b=0;b<3;b++)
for(c=0;c<3;c++)
a3d[a][b][c]=a+b+c;

for(a=0;a<3;a++)
 {
   printf("\n");
   for(b=0;b<3;b++)
    {
      for(c=0;c<3;c++)
      printf("%3d", a3d[a][b][c]);
      printf("\n");
     }
 }
getch();
}
```

Limitations of arrays

1. The compiler uses static memory allocation for an array that is it is not possible to increase or decrease the array size at runtime.
2. Elements cannot be inserted into an array.
3. We cannot delete elements into an array.
4. If the number of elements to be stored is not known in advance, there may be memory waste if an array of large size is specified.
5. If a small array size is specified there may not be enough memory to place all elements.
6. C does not perform bound checking of an array.

Difference between character array and integer array

Character array	Integer array
Character array NULL (\0) character is automatically added at the end.	Integer array or other types of arrays no NULL character is placed at the end.

→STRINGS: Declaration and Initialization of strings, String handling functions.

1. A group of characters can be stored in a character array. These character arrays are called strings.
2. To recognize a character array should end with a NULL character („\0‟).

Example: The string "student" could be stored as

„s‟	**„t‟**	**„u‟**	**„d‟**	**„e‟**	**„n‟**	**„t‟**	**„\0‟**

3. The length of a string is the number of characters it contains excluding NULL character. But, to store a string, we need one more locations than the length of the string.

4.14 Declaring strings:-The general form of declaration of a string variable is

Syntax: char variable-name [size];

Example: char name [10];

Scanf("%s", name);

The scanf( ) function usually accepts an address of the variable but in arrays ,by itself indicates the starting address of the array and hence we don‟t use an ampersand (&) before the variable name while reading strings.

4.15 Initializing strings:

Syntax: char name [size] ={ list of characters"};

Example: char name [25]= { „r‟ „a‟ „v‟ „i‟ , „\0‟ };

char name [ ]={ „r‟ „a‟ „v‟ „i‟ , „\0‟ };

char name [25]= "ravi";

char name [ ]= "ravi";

Note: While initializing a character array by listing its elements, must specify the NULL terminator („\0‟) with arrays elements.

Example: Write a C program to display the output when the count of NULL character in not considered.

```
#include<stdio.h>
#include<conio.h>
void  main()
{
char name[6]={'w','e','l','c','o','m'};
printf("name=%s", name);
getch();
}
```

Output: welcome followed by garbage characters

Explanation: The output of the above program would be welcome followed by some garbage values. To got the correct result the argument must be [7] instead of [6]. The output can be seen as given bellow after changing the argument [7] in place of [6].

Example: Write a C program to print "welcome" by using different formats of initialization of array.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char a1[9]={'w','e','l','c','o','m','e','\0'};
char a2[9]="welcome";
char a3[9]={ {'w'},{'e'},{'l'},{'c'},{'o'},{'m'},{'e'} };
clrscr();
printf("\n a1=%s",a1);
printf("\n a2=%s",a2);
printf("\n a3=%s",a3);
getch();
}
```

Example: Write a C program to display the string "prabhakar" using various format specifications.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char a1[15]="prabhakar";
clrscr();
printf("\n %s",a1);
printf("\n %.5s",a1);
printf("\n %.8s",a1);
printf("\n %.15s",a1);
printf("\n %-10.4s",a1);
printf("\n %11s",a1);
getch();
}
```

Output:

```
Prabhakar
Prabh
Prabhaka
prabhakar
prab
Prabhakar
```

4.16 Reading the strings: There are three ways to read to a string from the user through the keyboard. They are

1. By using scanf ( ) function 2.By using gets ( ) function 3.By using loops

1. By using scanf ( ) function: The scanf( ) function with format string %s can be used to read a string from the user.

Example:

```
void main()
{
char name[10];
scanf("%s", name);
getch();
}
```

The advantage of using scanf() function is that it can be used to read more than one string at a time.

Example: scanf("%s %s %s", name ,designation, address);

The disadvantages of scanf ( ) is, it terminate when it encounters a blank space.

2. By using gets ( ) function: The gets ( ) function overcomes the disadvantages of scanf( ) function, since it can read a string of any length with any number of blank spaces and tabs . It gets terminated only when an "ENTER" key is pressed.

Example:

```
void main()
{
char name[10];
gets(name);
getch();
```

}

The disadvantages of gets( ) function is that it can be used to read only a single string at a time.

3.By using the loops: A string is an array of characters, hence a string can also be read, character-by-character from the user by using loops.

```
void main()
{
char name[10];
for(i=0;i!='\0';i++)
scanf("%c", &name[i]);
getch();
}
```

Note: Ampersand (&) is used before the array name if we are reading a string character-by-character.

4.17 Writing strings: The puts( ) function writes the character string is supplied as a parameter to the standard output device.

Example: puts(name);

4.18 String library functions:

Various library functions in string.h

Function	**Purpose**	**Example**	**Result**
Strupr ( )	To convert all alphabets in string to upper case letters	Strupr(“srist”)	SRIST
Strlwr()	To convert all alphabets in string to lower case letters.	Strlwr(“SRIST”)	Srist

Strlen()	Finds the length of the string in bytes excluding NULL character	Char s[]=”city”;
		Int n;
		n=strlen(s);
			4

Strrev()	To reverse a string	Strrev(“city”);	Ytic

Strcpy()	Copies string str2 to string str1	Char s2[]=”city”
		Char s1[20];
		Strcpy(s1,s2);
			S2 contains city

Strcmp()	Used to compare if two strings are identical	n=strcmp(“srist”, “srist”)	n=0
Strncmp()	Compares the first n characters of s1 and s2	n=strncmp(“raj”,”ram”,2);	n=0
Strcat()	To join s2 to s1	Strcat(“stu”, “dent”)	student

Example 1: Write a C program to find the length of a given string

```
#include<strig.h>
void main()
{
char str[10];
printf("Enter the string");
gets(str);
printf("The length of a given string is %s", strlen(str));
getch();
}
```

Example 2: Write a C program to copy from one string into another string

```
void main()
{
char s1[10];
char s2[20];
printf("Enter the string");
gets(s1);
printf("The copied string is %s",strcpy(s2,s1));
getch();
}
```

Example 3:Write a C program to perform string concatenation

```
void main()
{
char s1[10];
char s2[20];
printf("Enter the string 1");
gets(s1);
printf("Enter the string 2");
gets(s2);
printf("The copied string is %s", strcat(s1,s2));
getch();
}
```

Example 4: Write a C program to reverse the given string

```
void main()
{
char  s1[10];
printf("Enter the string 1");
gets(s1);
printf("The reverse string is %s", strrev(s1));
getch();
}
```

Example 5: Write a C program to convert from upper to lower case

```
void main()
{
char  s1[10];
printf("Enter the string 1");
gets(s1);
printf("The lower case string is %s", strlwr(s1));
getch();
}
```

Example 6: Write a C program to convert from lower to upper case

```
void main()
{
char  s1[10];
printf("Enter the string 1");
gets(s1);
printf("The upper case string is %s", strupr(s1));
getch();
}
```

Example 7: Write a C program to check the strings are equal or not

```
void main()
{
char s1[10];
char s2[10];
printf("Enter the string 1");
gets(s1);
printf("Enter the string 2");
gets(s2);
x=strcmp(s1,s2);
if(x==0)
printf("strings are equal");
else
printf("strings are not equal");
getch();
}
```

Example: Write a C program to perform the string operations using string library functions.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
 char s[20],s2[20];
int x,ch;
clrscr();
printf("Enter the two strings");
scanf("%s %s",s,s1);
do
{
printf("\n   1.strlen");
printf("\n  2.strupr");
printf("\n  3.strlwr");
printf("\n  4.strrev");
printf("\n   5.strcat");
printf("\n  6.strcpy");
printf("\n 7.strcmp");
printf("\n 8.exit");
printf("\n Enter the choice");
scanf("%d",&ch);
switch(ch)
{
  case 1:printf("The length of the string is %d",strlen(s));
         break;
case 2:printf("The upper case of the string is %s",strupr(s));
         break;
case 3:printf("The lower case of the string is %s",strlwr(s));
```

```
        break;
case 4:printf("The reverse of the string is %s",strrev(s));
        break;
case 5:printf("The string concatination is %s",strcat(s,s1));
        break;
case 6:printf("The copieds string is %s",strcpy(s,s1));
        break;
case 7:if(x==strcmo(s,s1))
printf("The strings are equal);
else
printf("The strings are not equal");
        break;
case 8:exit(0);
default: printf("Invalid choice");
}
printf("Do u want continue press(y/n) buttons");
ch=getch();
}
while(ch=='y');
getch();
}
```

Example: Write a C program to find the length of a given string by using user defined functions

```
#inlcude<stdio.h>
#include<conio.h>
void main()
{
char str[10];
int i;
printf("Enter the string");
gets(str);
for(i=0;str[i]!='\0'; i++);
printf("The length of a string is %d", i);
getch();
}
```

Example: Write a C program to perform string copy by using user defined functions

```
#inlcude<stdio.h>
#include<conio.h>
void main()
{
char str[10],str1[10];
int  i;
printf("Enter the string");
gets(str);
for(i=0;str[i]!='\0'; i++)
{
 str1[i]=str[i];
}
str1[i]='\0';
printf("The copied string is %s",str1);
getch();
}
```

Example: Write a C program to perform string concatenation by using user defined functions

```
#inlcude<stdio.h>
#include<conio.h>
void main()
{
char str[10],str1[10];
int I, j;
printf("Enter the string 1");
gets(str);
printf("Enter the string 2");
gets(str1);
for(i=0;str[i]!='\0'; i++);
for(j=0;str1[j]!='\0';j++,i++)
{
 str[i]=str1[j];
}
str[i]='\0';
printf("The string concatenation is %s", str);
getch();
}
```

Example: Write a C program to reverse the given string by using user defined functions

```
#inlcude<stdio.h>
#include<conio.h>
void main()
{
char   str1[10],rev[10];
int  I, j ,n=0;
printf("Enter the string");
gets(str1);
for(i=0;str[i]!='\0'; i++)
n++;
for(j=n-1,i=0;j>=0;j--,i++)
{
 rev[i]=str1[j];
}
rev[i]='\0';
printf("The reverse string is%s", rev);
getch();
}
```

Example: Write a C program to check whether the strings are equal or not by using user defined functions.

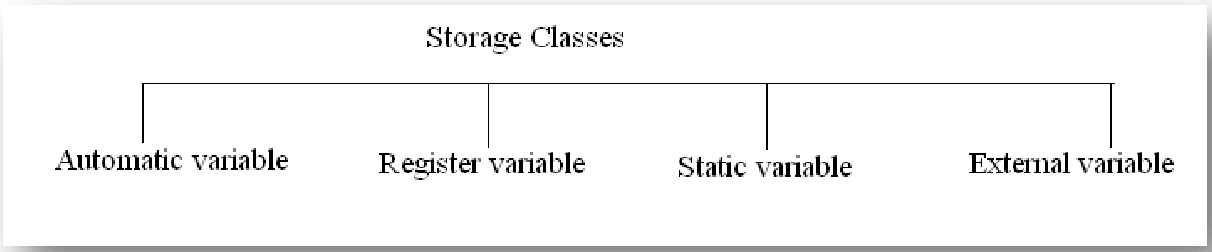
```
#inlcude<stdio.h>
#include<conio.h>
void main()
{
char str1[10],str2[10];
int i,j,flag;
printf("Enter the string1");
gets(str1);
printf("Enter the string2");
gets(str2);
for(i=0;str1[i]!='\0'; i++)
{
for(j=0;str2[j]!='\0';j++)
{
if((str1[i]!=str2[j])&&(i==j))
{
```

```
flag=1;
}
}
}
if((flag==1)||(i!=j))
printf("The strings are not equal");
else
printf("The strings are equal");
getch();
}
```

Example: Write a C program to check whether the string is palindrome or not.

```
#inlcude<stdio.h>
#include<conio.h>
void main()
{
char  str[10];
int i, j,flag=0, n;
printf("Enter the string");
gets(str);
n=strlen(str);
for(i=0, j=n-1; str[i]!='\0'; i++, j--)
{
if(str[i]!=str[j])
{
flag=1;
break;
}
}
if((flag==0)
printf("The string is Palindrome");
else
printf("The string is not palindrome");
getch();
}
```

→**STORAGE CLASSES:** Automatic, External, Static and Register Variables.



Normally the life of a variable is limited to a function as long as the function is alive. How to make it alive in a file or throughout the program or limiting only to a block inside a function or to make common to a desired couple of functions etc., The answer lies in “STORAGE CLASSES” or “VARIABLE TYPES”.

There are four types of storage classes.

1. Automatic variables. 2. External variables. 3. Static variables. 4. Register variables.

1. Automatic variables:

Automatic variables are declared inside a function, in which they are to be utilized. They are created when the function is called and destroyed automatically when the function is exited. By default all variables are automatic variables. These are also called local or internal variables.

Ex. Write a C Program on Automatic Variables.

```
#include<stdio.h>
#include<conio.h>
Void main()
{
int a=9;
clrscr();
fun1();
printf(“\n in main() a=%d”,a);
getch();
}
fun1()
{
Auto int a=8;
fun2()
{
int a =7;
printf(“\n in fun2() a=%d”,a);
}
```

Output:

```
In fun2 () a=7
In fun1 () a=8
In main () a=9
```

2. Register variables:

1. This is local to a function or a block.
2. If a compiler finds a physical register in the CPU free for the time being, and also big enough to hold the value, then it may stick that variable in that register. Otherwise the compiler treats that variable as ordinary.
3. It is machine dependent. But compiler will not give error messages even if no register available in reserve.

Ex: Write a C program on Register variables.

```
#include<stdio.h>
#include<conio.h>
Void main()
{
register int a=1;
clrscr();
for(    ;a<=3;a++)
printf("\na=%d",a);
getch();
}
```

Output:

```
a=1        a=2                  a=3
```

3. Static variables:

When a variable is declared as a static variable it is assigned the value zero. Static variables are initialized only once. They will not be initialized for second time during the program. When static is applied to a global variable, the global variable becomes inaccessible outside the file.

Ex: Write a C program on Static Variables.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a;
static int b;
clrscr();
printf("\na=%d\nb=%d",a,b);
getch();
}
```

Output:

```
a=25338,          b=0;
```

4. External variables:

Variables that are both alive and active throughout the entire program are called external variables. These variables are available to all functions in that program. Whatever changes that occur in a function, will affect the value of that variable.

Ex: Write a C program on External Variables.

```
#include<stdio.h>
#include<conio.h>
int a=5;
void main()
{
clrscr();
fun1();
fun2();
printf("\n In main() a=%d",a);
getch();
}
fun1()
{
printf("\n in fun1() a=%d",a);
}
fun2()
{
Printf("\n in fun2() a=%d",a);
}
```

Output:

```
In fun1 () a=5          In fun2 () a=5          In main () a=5
```

Storage class	Keyword	Storage place	Scope of the variable	Initialization variable	Type of the variable
Automatic variable	Auto	Memory	Inside the block in which it is defined	Garbage value	local
Register variable	Register	Cpu register	Inside the block in which it is defined	Garbage value	Local
Static variable	Static	Memory	The value of the variable persists between different function call.	Zero	Local
External variable	Extern	Memory	The value of the variable persists as long as the program execution comes to an end.	Zero	global