```
strstr(s1,s2);
```

Example : If s1="college" and s2="eg",

strstr(s1,s2);

Output will be "ege".

If s1="college" and s2="l",

strstr(s1,s2);

Output will be "llege".

14. strchr()

Returns a pointer to the first occurrence of the character in string. This takes the forms:

```
strchr(s1,c);
```

Example:

```
#include <string.h>
#include <stdio.h>
#include<conio.h>
void main()
{
char s1[100],c,*ptr;
gets(s1);
fflush(stdin);
scanf("%c",&c);
ptr = strchr(s1, c);
if (ptr)
   printf("The character %c is at position: %d", c, ptr-s1);
else
   printf("The character was not found");
getch();
 }
```

Output:

```
College                          college
o                                e
The character o is at position: 1        The character e is at position: 4
```

■ PRACTICE PROGRAMS IN ARRAY

1. A program to sort ten different numbers provided by the users in descending order.

```
#include<stdio.h>
#include<conio.h>

void main()
```

```
{
    int n[10],i,j,temp;

    clrscr();
    for(i=0; ; i<=9 ; i++)
    {
        printf("Enter element %od : ",i+1);
        scanf("%od",&n[i]);
    }

    for(i=0 ; i<=9 ; i++)
    {
        for(j=i ; j<=9 ; j++)
        {
            if( n[i] < n[j] )
            {
                temp = n[j];
                n[j] = n[i];
                n[i]=temp;
            }
        }
    }

    printf("\n\nThe SORTED ARRAY is : \n");
    for(i=0 ; i<=9 ; i++)
    {
        printf("\t\t%d\n",n[i]);
    }

    getch();
}
```

Output

```
Enter element 1 : 1
Enter element 2 : 43
Enter element 3 : 22
Enter element 4 : 34
Enter element 5 : 54
Enter element 6 : 66
Enter element 7 : 67
Enter element 8 : 89
Enter element 9 : 81
Enter element 10 : 97
The SORTED ARRAY is:
97  89 81 67 66 54433422    1
```

2. Program to display Sum and average of ten values.

```
#include<stdio.h>
#include<conio.h>

void main()
{
   int a[10],i,SUM;
float AVG;
   clrscr();
SUM=0; AVG=0;
printf("Enter 10 values in array\n\n");
   for(i=0 ; i<=10 ; i++)
   {
      scanf(" %d : ",&a[i]);
   }

   for(i=0 ; i<=10 ; i++)
   {
      SUM = SUM + a[i];
   }
   printf("\n\n Sum of Array Values= %d", SUM);
   AVG = SUM/10.0;
   printf("\n\n Average of Array= %f", AVG);
getch();
}
```

Output

```
Enter 10 values in Array:
2
4
5
6
7
8
9
2
5
6
Sum Of Array Values=54
Average of Array values=5.400000
```

3. **Program to find maximum number in Array.**

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[10], i, MAX, temp;
clrscr();
max=0;
printf("Enter 5 values in Array \n\n");
for(i=0; i<=5; i++)
{
scanf("%d", &a[i]);
}
for(i=0; i<=5; i++)
{
if(a[i]>MAX)
{
temp=a[i];
MAX=temp;
}
}
printf("The Maximum Number in Array=%d", temp);
getch();
}
```

Output

```
Enter 10 values in Array
2
4
5
6
7
The Maximum Number in Array: 7
```

4.6 INTRODUCTIONS TO POINTERS

A pointer is a variable that points to a memory location at which data is stored. Each memory cell in the computer has an address that can be used to access that location so a pointer variable points to a memory location we can access and change the contents of this memory location via the pointer.

In C any declared variable has an appropriate location in the memory. Each location has unique address and addresses are sequential. Variable requires storage space in memory depending on its type. For example, char type requires 1 byte, int type requires 2 bytes, float type requires 4 bytes, double type requires 8 bytes.

```
int a=123;
```

Here, the value 123 is stored at the location of **int** type variable **a** in the memory. Assume that the address of the location is 1000. Then it can be represented as

Now, if the address of variable **a** (**1000**) is stored in another variable **p**, then the variable **p** is known as pointer to variable **a**, because the variable **p** will hold the address of **a** (**1000**) as value. It can be represented as

To assign/access the address of variable, the '**address of' operator** (**&**) is used. For example, address of variable **a** can be assigned to pointer variable **p** as

```
p = &a;
```

So, here the address of variable **a** (**1000**) will be assigned to pointer variable **p**.

To assign/access the value of variable using pointer, the *** operator**('**value at' operator** or **dereferencing operator** or **indirection operator**') is used. For example, value of variable **a** (123) can be accessed using pointer **p** as :

```
int val=*p;
```

So, here the *p (value at p, but **p=&a**, so value at **&a** means value at address of **a**) will assign 123 (value at **a**) to the variable **val**. So, *p and **a** represents the same value, i.e. **123** (value of **a**).

4.7 CHARACTERISTICS OF POINTERS

A pointer is a variable.

A pointer stores an address of another variable.

The type of pointer indicates the type of variable the pointer is pointing to.

Pointers are used to manipulate data using the address of variables.

Pointers allow us to access low level memory.

The pointer accessing method is faster than array indexing.

Dynamic memory allocation is done using pointers.

Pointers are the only way to do memory level computations.

Pointers produce compact and efficient code.

Pointers provide a very powerful programming tool.

4.8 THE ADDRESS-OF OPERATOR (&) AND INDIRECTION OPERATOR (*)

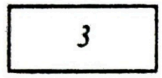
4.8.1 The Address-of Operator (&)

The C language provides an operator, &, to get the address of a variable. & operator is called the address-of operator because it can return the address of a variable. To assign/access the address of variable, the 'address of' operator (&) is used.

For example:

```
int i=3;
```

Meaning:-



6021

6024 any arbitrary address.

For example, address of variable a can be assigned to pointer variable p as:

```
int a, *p;
p = &a;
```

So, here the address of variable a (=1000) will be assigned to pointer variable p.

The following code, for example,

```
long int x, *y;
y = &x;
```

assigns the address of the long integer variable x to another variable, y.

Program to get the address of variable:

```
/* Obtaining addresses of variables*/
#include <stdio.h>
int main()
{
char c;
int x;
float y;
printf("c: address=0x%p, content=%c\n", &c, c);
printf("x: address=0x%p, content=%d\n", &x, x);
printf("y: address=0x%p, content=%5.2f\n", &y, y);
c = 'A';
```

```
x = 7;
y = 123.45;
printf("c: address=0x%p, content=%c\n", &c, c);
printf("x: address=0x%p, content=%d\n", &x, x);
printf("y: address=0x%p, content=%5.2f\n", &y, y);
return 0;
}
```

Output:

```
c: address=0x1AF4, content=@
x: address=0x1AF2, content=-32557
y: address=0x1AF6, content=0.00
c: address=0x1AF4, content=A
x: address=0x1AF2, content=7
y: address=0x1AF6, content=123.45
```

Variable	x		c		y			
Address	1AF2	1AF3	1AF4	1AF5	1AF6	1AF7	1AF8	1AF8
Value before	–32557		@		0.00			
Value after	7		A		123.45			

4.8.2 Indirection operator (*)

It's the asterisk ('*') operator. So, if you have a pointer:

```
int x;
int *p = &x;
```

you can set x indirectly through p this way:

```
*p = 5;
```

The unary * operator, applied to a pointer, is the indirection operator.

4.9 DECLARATION AND INITIALIZATION OF POINTERS

Pointers can be declared and initialized in same way as other variables.

4.9.1 Pointer Declaration

The general format of pointer variable declaration is:

```
data_type *pointer_name;
```

Example:

```
int *ptr;
float *fl;
```

Here, ptr is a pointer variable which points to the integer type variable. And fl points to the floating-point variable.

4.9.2 Pointer Initialization

After declaring, the pointer can be initialize using & operator.

Example:

```
int a=3,b;
int *ptr;
ptr = &a;
```

Here, **ptr** is a pointer which points to the integer type variable a. Means address of variable **a** is stored in **ptr**.

Pointer can be initialized at the time of declaration also:

```
int a=3,b;
int *ptr = &a;
b=*ptr;
```

Here, address of variable **a** is stored in **ptr**. And ***prt** (value at **ptr**, means value at **&a**, means **3**) is assigned to **b**, so **b=3**.

Example

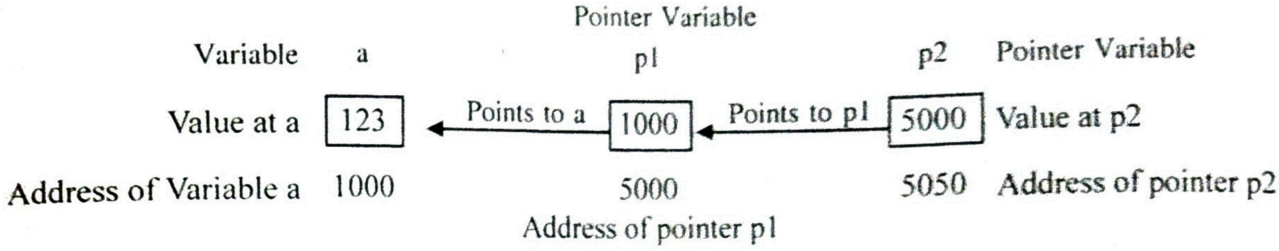
```
#include <stdio.h>
#include<conio.h>
void main()
{
int a=3,b=8,*aptr,*bptr=&b;
clrscr();
aptr=&a;
printf("a = %d\n",a);
printf("&a = %d\n",&a);
printf("b = %d\n",b);
printf("&b = %d\n",&b);
printf("*aptr = a = %d\n",*aptr);
printf("&*aptr = &a = %d\n",&*aptr);
printf("*bptr = %d\n",*bptr);
printf("&*bptr = &b = %d\n",&*bptr);
getch();
}
```

Output:

```
a = 3
&a = -12
b = 8
&b = -14
*aptr = a = 3
&*aptr = &a = -12
*bptr = b = 8
&*bptr = &b = -14
```

4.10 POINTER TO POINTER (CHAIN OF POINTER)

Pointer to pointer means pointer points to another pointer. Pointer contains the address of another pointer variable of specific data type. It can be represented as:



Syntax:

```
data_type  **pointer_name;
```

Example:

```
#include <stdio.h>
#include<conio.h>
void main()
{
        int a=3,*pt,**ptr;
        clrscr();
        pt=&a;
        ptr=&pt;
        printf("&a = %d\n",&a);
        printf("a = %d\n",a);
        printf("&pt = %d\n",&pt);
        printf("pt = &a = %d\n",pt);
        printf("*pt = a = %d\n",*pt);
        printf("&ptr = %d\n",&ptr);
        printf("ptr = &pt = %d\n",ptr);
        printf("*ptr = pt = &a = %d\n",*ptr);
        printf("**ptr = *pt = a = %d\n",**ptr);
        getch();
}
```

Output:

```
&a = -12
a = 3
&pt = -14
pt = &a = -12
*pt = a = 3
&ptr = -16
ptr = &pt = -14
*ptr = pt = &a = -12
**ptr = *pt = a = 3
```

```
        p=x;
        for(i=0;i<5;i++)
        {
            sum=sum+*p;
            p++;
        }
        printf("Sum of all Elements = %d",sum);
        getch();
}
```

Output:

```
Sum of all Elements = 30
```

Pointer Vs Array

Pointer	Array
– A pointer variable is a place in memory that keeps address of another variable.	– An array is a group of same types of continuous fixed memory locations.
– When a pointer is declared, the compiler allocates memory for storing the address of another variable only at runtime.	– When an array is declared, the compiler allocates a base address and sufficient amount of storage to contain all the elements of the array in contiguous memory locations.
– The pointer contains address of different variables at different time in execution.	– Initially the array contains the base address as the location of the first element (index 0) of the array.
– Pointer can contains address of only one variable at any time.	– An array contains addresses of all its elements.
– When we allocate memory for a pointer to use it as a dynamic array. The memory can be resized or freed later.	– This is not the case for arrays.
– The pointer accessing method is faster than array indexing.	– The array indexing to access elements is slower.
– Pointers are used to manipulate data using the address of variables.	– Arrays are used to manipulate data using the indexes of various elements.
– Pointers use * operator to access the data pointed to by them.	– Arrays use [ ] brackets operator and index of elements to access the data of the element.

4.11.1 Pointers and Character Strings

We know that a string is an array of characters, terminated with null characters. Like a one-dimensional array we can use a pointer to access the individual characters in a string.

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
char name[10]="DELHI";
int length;
char *cptr=name;
clrscr();
while(*cptr!='\0')
{
printf("%c is stored at address %u\n",*cptr,cptr);
cptr++;
}
length=cptr-name;
printf("Length of string : %d",length);
getch();
}
```

Output:

```
D is stored at address 65516
E is stored at address 65517
L is stored at address 65518
H is stored at address 65519
I is stored at address 65520
Length of string : 5
```

4.11.2 Array of Pointer

Array of pointer is the collection of pointers of same type, means collection of addresses of same type variable.

Syntax:

```
data_type  *pointer_name[size];
```

For Example:

```
int *ptr[5];
int a[5][5];
for(i=0;i<5;i++)
{
ptr[i]=&a[i][0];
}
```

Here, **ptr** is an array of pointer of int type, means it contains addresses of int type.