Module 4- Strings and pointers

- A group of characters together is called string.
- String is always enclosed within double quotes(“:”)
- String always ends with deli meter (NULL,’\0’).

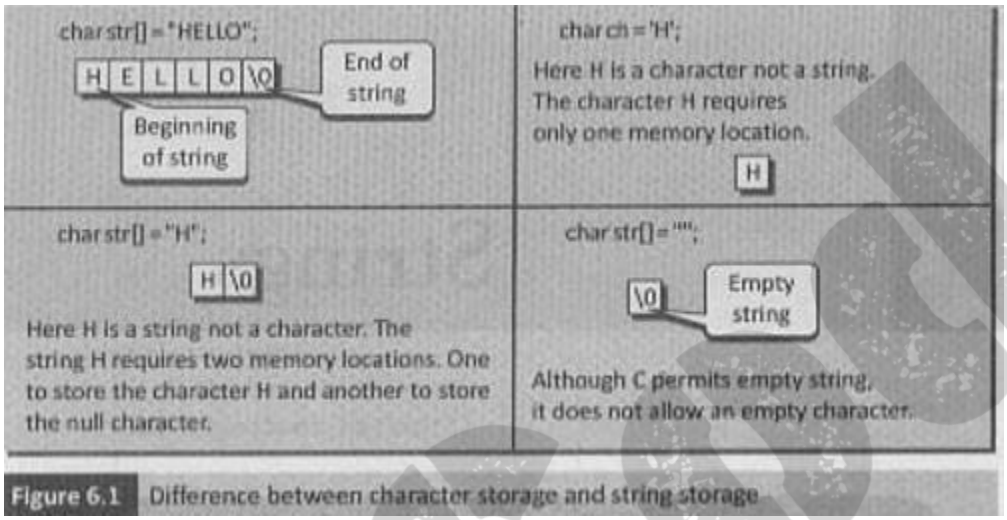


Figure 6.1 Difference between character storage and string storage

Declaration of a string:

A string is declared like an array of character.

- **Syntax:** data type string name [size];
- **Example:** char str [10];

Initialization of a string:

Syntax: data type string name [size]=value;

Example:

1. Char str []={‘H’, ’e’, ’l’, ’l’, ’o’, ’\0’};
   Here compiler will automatically calculate the size based on the number of elements initialized.so,in this example 6 memory slots will be reserved to store the string variable.

2. Char str [10]=”HELLO”;
   The compiler created a character array of size 10, stores the value “HELLO” in it, and finally terminates the value with a null character. Rest of the elementin the array is automatically initialized to NULL.

3. Char str[5]=”HELLO”;

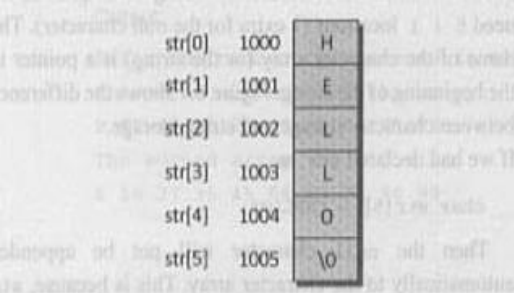


Figure 6.2 Memory representation of a character array

Reading strings:

If we declare a string by writing

Char str[100];

Then str can be read from the user by using three ways:

1. Using scanf function.
2. Using gets () function.
3. Using getchar() ,getch() or getche() function repeatedly.

The string can be read using scanf() by writing

Scanf(“%s”,str);

The main drawback with this function is that it terminates as soon as it finds a blank space.

Example: if the user enters “Hello world”, then str will contain only “Hello”. This is because the moment a blank space is encountered, the string is terminated by the scanf() function.

The next method of reading a string is by using gets() function.

Syntax: gets(str);

gets() is a simple function that overcomes the drawbacks of scanf().The gets() function takes the starting address of the string which will hold the input .

Example:

```
i=0;
getchar(ch); // Get a character
while(ch != '*')
{
   str[i] = ch;
      // Store the read character in str
   i++;
   getchar(ch); // Get another character
}
str[i] = '\0';
   // terminate str with null character
                  str[i] = '\0';
```

Writing strings:

The strings can be displayed on screen using three ways:

1. Using printf() function.
2. Using puts() function.
3. Using putchar() function repeatedly.

The string can be displayed using printf() by writing

Printf("%s",str);

The next method of writing a string is by using the puts function.The string can be displayed by writing

Puts(str);

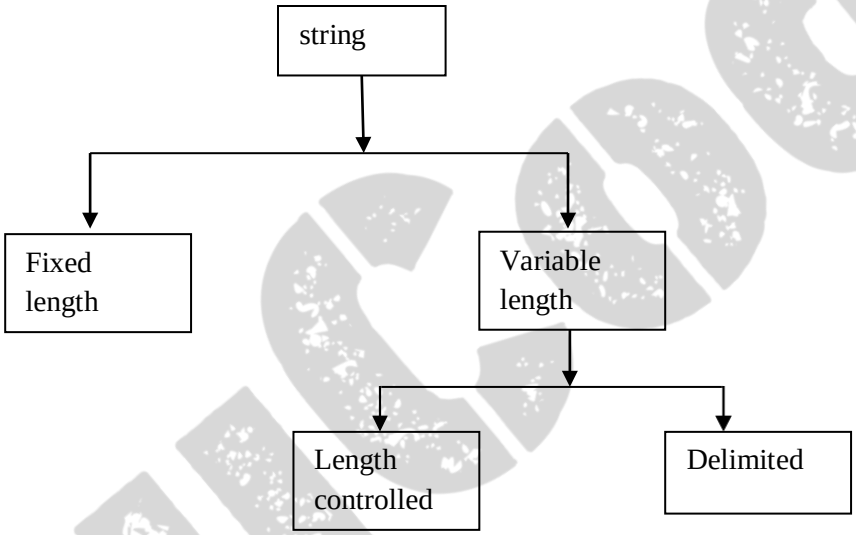
Puts() is a simple function that overcomes the drawbacks of printf ().the puts() function writes a line of output on the screen. It terminates the line with a newline character ('\n').it returns an EOF(end of file)(-1)if an error occurs and returns a positive number on success.

Example program:

```
i=0;
while(str[i] != '\0')
{
   putchar(str[i]);
      // Print the character on the screen
   i++;
}
```

String taxonomy:

In c we can stor ea string either in fixed length format or in variable length format.



Fixed length string:

When storing a string in a fixed length format, you need to specify an appropriate size for the string variable. If the size is too small, then you will not be able to store all the elements in the string. On the other hand, if the string size is large, then unnecessarily memory space will be wasted.

Variable length string:

The string can be expanded or contracted to accommodate the elements in it.

Example: Declare string variable to store the name of a student. If student has a long name of say 20 characters, then the string can be expanded to accommodate 20 characters, on the other

hand, a student name has only 5 characters, then the string variable can be contracted to store only 5 characters.

Length –controlled string:

In length controlled string you need to specify the number of characters in the string. This count is used by string manipulation function to determine the actual length of the string variable.

Delimited string:

In this format the string is ended with a delimiter. The delimiter is then used to identify the end of the string.

For example in English language every sentence is ended with a full-stop, similarly in C we can use any character such as comma, semicolon, colon, dash, null character etc. as the delimiter of a string.However,the null character is the most commonly used string delimiter in the C language.

String operations:

1. **Length:**
   The number of characters in the string constitutes the length of the string.
   Example: Length ("C programming is fun")
   Output: return 20

   Example program:

```
#include<stdio.h>
#include<conio.h>
int main()
{
    Char str[100],i=0,length;
    Clrscr();
    Printf("\n enter the string:");
    gets(str);
    while(str[i]!='\0')
    i++;
    length=I;
   printf("\\n the length of the string is:%d",length);
   getch();
}
```

Converting characters of a string into upper case:

- We have already seen that in memory the ASCII codes are stored instead of the real value. The ASCII code for A-Z varies from 65 to 91 and ASCII code for a-z ranges from 97-123.
- If we have to convert a lower case character to upper case then we just need to subs tract 32 from the ASCII value of the character.

Example Program:

```
#include<stdio.h>
#include<conio.h>
int main()
{
        Char str[100],upper_str[100];
         int i=0,j=0;
         printf("\n enter the string:");
         gets(str);
         while(str[i]!="\0")
{
   if(str[i]>="a" && str[i]<="z")
  upper_str[j]=str[i]-32;
 else
 upper_str[i]=str[i];
 i++;
 j++;

}

  Upper_str[j]="\0";

 Printf("\n the string converted into upper case is:");

 Puts(upper_str);

 return 0;

}
```

Output:

enter the string : hello
The string converted into lower case is : HELLO

Converting characters of a string into lower case:

- The ASCII code for A-Z varies from 65 to 91 and the ASCII code for a-z ranges from 97-123.
- If we have to convert an upper case character into lower case, then we just need to add 32 to its ASCII value.

Example Program:

```
#include<stdio.h>
#include<conio.h>
int main()
{
        Char str[100],upper_str[100];
         int i=0,j=0;
         printf("\n enter the string:");
         gets(str);
         while(str[i]!="\0")
{
    if(str[i]>="a" && str[i]<="z")
    lower_str[j]=str[i]-32;
    else
    lower_str[i]=str[i];
    i++;
    j++;
}
 lower_str[j]="\0";
 Printf("\n the string converted into lower case is:");
 Puts(lower_str);
 return 0;
}
```

Output:

```
Enter the string :Hello
The string converted into lower case is :hello
```

Concatenating two strings to form a new string:

If s1 and s2 are two strings, then concatenation operation produces a string which contains characters of s1 followed by the characters of s2.

Example Program:

```
#include<stdio.h>
#include<conio.h>
int main()
{
        Char str1[100],str2[100],str3[100];
        int i=0,j=0;
        printf("\n enter the first string :");
        gets(str1);
       printf("\n enter the second string:");
       gets(str2);
      while(str1[i]!='\0')
{
      Str3[j]=str1[i];
      i++;
      j++;
}
       i=0;
     while(str2[i] !='\0')
{
     Str3[j]=str2[i];
     i++;
     j++;

}
   Str3[j]='\0';
   Printf("\n the concentrated string is:");
   Puts(str3);
   getch();
   return 0;
}
```

Output:

Enter the first string : Hello

Enter the second string : How are you?

The concentrated string is : Hello,How are you?

Appending strings:

- Appending one string to another involves copying the contents of the source string at the end of the destination string.
- Example: if s1 and s2 are two strings, then appending s1 to s2 means we have to add the contents of s1 to s2.
- Here s1 is the source string and s2 is the destination string. The appending operation would leave the source string s1 unchanged and the destination string s2=s2+s1.

Example program:

```
#include <stdio.h>

#include<conio.h>

main()

{

        Char Dest_str[100],source_str[50];

        int i=0,j=0;

      printf("\n enter the source string:");
      gets(source_str);
      printf("\n enter the destination string:");
      gets(Dest_str);
      while(Dest_str[i]!='\0')
      i++;
      while(source_str[j]!='\0')

{

    Dest_str[i]=source_str[j];

    i++;
```

```
    j++;
}
        Dest_str[i]='\0';
        Printf("\n After appending , the destination string is :");
       Puts(Dest_str);
       getch();
       return 0;
}
```

Output:

Enter the source string :How are you?

Enter the destination string:Hi,

After appending,the destination string is :Hi,How are you?

Comparing two strings:

If s1 and s2 are two strings then comparing two strings will give either of these results:

1. S1 and S2 are equal
2. S1> S2,when in dictionary order s1 will come after s2.
3. S1<S2 ,when in dictionary order s1 precedes s2.

Example program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
        Char str1[50],str2[50];
        int i=0,len1=0,len2=0,same=0;
```

```
        printf("\n enter the first string :");
        gets(str1);
        printf("\n enter the second string ");
       gets(str2);
len1=strlen(str1);
len2=strlen(str2);
if(len1==len2)
{
          While(i=len1)
{
     if(str1[i]==str2[i])
     i++;
    else
    break;
}
if(i==len1)
{
     Same=1;
     Printf("\n the two strings are equal");
}
}
if(len1!=len2)
printf("\n the two strings are not equal ");
if (same==0)
{
```

```
if(str1[i]>str2[i])
printf("\n string1 is greater than string2");
else if(str1[i]<str2[i]))
printf("\n string2 is greater than string1");
}
getch();
return 0;
}
```

Output:

Enter the first string: Hello

Enter the second string: Hello

The two strings are equal

Reversing a string:

If s1="Hello", then reverse of s1="olleH".To reverse a string we just need to swap the first character with the last. Second character with the second last character and so on.

Note:

There is a library function strrev (s1) that reverses all the characters in the string except the null character. It is defined in string.h.

Example:

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

```
  Char str[100],reverse_str[100],temp;
 int i=0,j=0;
 printf("\n enter the string");
 gets(str);
 j=strlen(str)-1;
while(i<j)
{
  temp=str[j];
 str[j]=str[i];
 str[i]=temp;
 i++;
  j--;
}
Printf("\n the reversed string is :");
Puts(str);
getch();
return 0;
}
```

Output:

Enter the string :Hi there

The reversed string is:ereht iH

Extracting a substring from the left of a string:

In order to extract a substring from the main string we need to copy the content of the string starting from the first position to the nth position where n is the number of characters to be extracted.

Example:

If s1="Hello world",then substr_left(s1,7)=Hello w

Extracting a substring from the right of a string:

In order to extract a substring from the right side of the main string we need to first calculate the position.

Example:

If s1="Hello world" ,then substr_right(s1,7)="o world"

Extracting a substring from the Middle of a string:

To extract a substring from a given string requires information about three things.The main string the position of the first character of the substring in the given string and maximum number of characters/length of the substring.

Example:

Str[]="Welcome to the world of programming";
Then, Substring(str,15,5)=world

Insertion:

The insertion operation insert a string S, in the main text T, at the K^{th} position.The general syntax of this operation is : INSERT(text,position,string).

Example: INSERT("xyzxyz",3,"AAA")="xyzAAAxyz".

Program:

```
#include<stdio.>
#include<conio.h>
main()
{
Char text[100], str[20], ins_text[100];
int i=0,j=0,k=0,pos;
Printf("\n enter the main text:");
gets (text);
printf("\n enter the string to be inserted:");
```

```
gets(str);
printf("\n enter the position at which the string has to be inserted:"");
scanf("%d",&pos);
ehile(text[i]!='\0')
{
        if(i==pos)
{
        While(str[k]!='\0')
{
    ins_text[j]=str[k];
    j++;
    k++;
}
}
else
{
     ins_text[j]=text[i];
     j++;
}
i++;
}
   ins_text[j]='\0';
   printf("\n the new string is:");
   puts(ins_text);
   getch();
   return 0;
}
```

Output:

Enter the main text: How you?

Enter the string to be inserted: are

Enter the position at which the string has to be inserted: 6

The new string is: How are you?

Indexing:

Index operation returns the position in the string where the string pattern first occurs.

Example:

INDEX(“Welcome to the world of programming”,”world”)=15

Deletion:

The deletion operation deletes a substring from a given text.we write it as DELETE(text,position,length).

Example:

DELETE(“ABCDXXXABCD”,5,3)=”ABCDABCD”

Program:

```
#include<stdio.h>
#include<conio.h>
main()
{
    Char text[200],str[20],new_text[200];
     int i=0,j=0,found=0,k,n=0,copy_loop=0;
printf(“\n enter the main text:”);
gets(text);
fflush(stdin);
printf(“\n enter the string to be deleted:”);
gets(str);
fflush9stdin);
while(text[i]!=’\0’)
{
    j=0,found=0,k=i;
    while(text[k]==str[j] && str[j]!=’\0’)
    {
        k++;
```

```
        j++;
    }
if(str[j]=='\0')
        copy_loop=k;
        new_text[n]=text[copy_loop];
        i++;
        copy_loop++;
        n++;
        }
        new_str[n]='\0';
        printf("\n the new string is :");
        puts(new_text);
        getch();
        return 0;
}
```

Output:

Enter the main text :Hello,how are you?

Enter the string to be deleted :,how are you?

The new string is:Hello

Relpacement:

Replacement operation is used to replace the pattern p1 by another pattern t2 . This is done by writing,REPLACE(text,pattern1,pattern2)

Example:

("AAABBBCCC","BBB","X")=AAAXCCC

Miscellaneous string and character functions:

Function	Usage	Example
isalnum(int c)	Checks whether character c is an alphanumeric character	isalpha('A');
isalpha(int c)	Checks whether character c is an alphanumeric character	isalpha('z');
iscntrl(int c)	Checks whether character c is a control character	Scanf("%d",&c); iscntrl(c);
isdigit(int c)	Checks whether character c is a digit	isdigit(3);
isgraph()	Checks whether character c is a graphic or printing character.The function excludes the white space character	isgraph('!');
isprint(int c)	Checks whether character c is a printing character.the function includes the white space charcter	isprint('@');
Islower(int c)	Checks whether the character c is in lower case	Islower('k');
Isupper(int c)	Checks whether the character c is in upper case	Isupper('K');
Ispunct(int c)	Checks whether the character c is a white space character	Isspace('')
Isxdigit(int c)	Checks whether the character c is a hexadecimal digit	Isxdigi('F');
tolower(int c)	Converts the character c to lower case	tolower('K') Returns k
toupper(int c)	Converts the character c to upper case	Tolower('k') returns K

String Manipulation functions:

1. **Strcat() function:**

Syntax: char*strcat(char *str1,const char*str2);

The strcat() function appends the string pointed by str2 to the end of the string pointed to by str1.

Example:

#include<stdio.h>

#include<string.h>

```
int main()
{
    Char str1[50]=””Programming”;
    Char str2[]=”in c”;
    Strcat(str1,str2);
     Printf(“\n str1: %s”,str1);
     return 0;
}
```

Output:

Str1:Programming in c

2. strncat() function:

Syntax:
char *strncat(char *str1,const char *str2,size_t n);

This appends the string pointed to by str2 to the end of the string pointed to by str1 upto n characters long.

Example:

```
#include<stdio.h>
#include<string.h>
int main()
{
    Char str1[50]=”programming”;
    Char str2[]=”in c”;
    Strncat(str1,str2,2);
    Printf(“\n str1:%s”,str1);
   return 0;
}
```

Output:

Str1:programming in

3. Strchr() function:

Syntax:

Char*strchr (const char *str, int c);

This function searches for the first occurrence of the character c in the string pointed to by the argument str. The function returns a pointer pointing to the first matching character, or null if no match was found.

Example:

```
#include<stdio.h>
#include<string.h>
int main()
{
    Char str[50]=”programming in c”;
    Char *pos;
    Pos=strchr(str,’n’);
    if(pos)
    printf(“\n n is found in str at position %d”,pos);
   else
   printf(“”\n n is not present in the string”);
   return 0;
}
```

Output:

n is found in str at position 9

4. Strrchr()function:

Syntax:

Char*strrchr(const char *str,int c);

The strchr() function searches for the first occurrence of the character c beginning at the rear end and working towards the front in the string pointed to by the argument str.

Example:

```
#include<stdio.h>
#include<string.h>
int main()
{
    Char str[50]=”programming in c”;
    Char *pos;
     if(pos)
    printf(“\n the last position of n is :%d”,pos-str);
   else
   printf(“\n n is not present in the string”);
   return 0;
}
```

Output:

The last position of n is:13

5. strcmp() function:

Syntax:

int strcmp(const char * str1,const char *str2);

the strcmp compares the string pointed to by str1 to the string pointed to by str2.The function return zero if the strings are equal.Otherwise,it returns a value less than zero or greater than zero if str1 is less than or greater than str2 .

Example:

```
#include<stdio.h>
#include<string.h>
int main()
{
    Char str1[10]=”HELLO”;
    Char str2[10]=”HEY”;
    if(strcmp(str1,str2)==0)
   printf(“\n the two strings are identical”);
   else
   printf(“\n the two strings are not identical”);
return 0;
}
```

Output:

The two strings are not identical

}

6. **Strcpy() function:**

Syntax:

Char *strcpy(char *str1,const char *str2);

This function copies the string pointed to by str2 to str1 including the null character of str2.it returns the argument str1.

Example:

Example:

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

```
int main()
{
    Char str1[10]=”HELLO”;
    Char str2[10]=”HEY”;
   strncpy(str1,str2,2)
   printf(“\nstr1:%s”,str1);
   return 0;
}
```

Output:

HE

7. **strlen() function:**

Syntax:

Size_t strlen(const char *str);

This function calculates the length of the string str upto but not including the null character, i.e the function returns the number of characters in the string.

Example:

```
#include<stdio.h>
#include<string.h>
int main()
{
   Char str[]=”HELLO”;
    Printf(“\n length of str is :”%d”,strlen(str));
    return 0;
}
```

Output:

Length of str is :5

8. strstr() function:

Syntax:

Char*strstr(const char * str1,const char *str2);

The function is used to find the first occurrence of string str2 in the string str1. It returns a pointer to the first occurrence of str2 in str1.If no match is found then null pointer is returned.

Example:

```
#include<stdio.h>
#include<string.h>
int main()
{
    Char str1[]=”HAPPY BIRTHDAY TO YOU”;
     Char str2[]=”DAY”;
     Char *ptr;
     Ptr=strstr(str1,str2));
     if(ptr)
     printf(“\n substring found”);
    else
    printf(“\n substring not found”);
    return 0;
}
```

Output:

Substring found

Array of string:

- String is an array of characters.
- For example if we say, char name[]=”Mohan”, then name is a string(character array) that has five characters.
- Now suppose that there are 20 students in a class and we need a string that stores names of all the 20 students.Here we need a string of strings or an array of strings.Such an array of strings would store 20 individual strings.An array of sring is declared as,

Char name[20][30];

Here the first index will specify how many strings are needed and the second index specifies the length of every individual string.So here we allocate space for 20 names where each name can be a maximum of 30 characters long.

Syntax:

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

Memory representation of an array of strings

Char name[5][10]={“Ram”,”Mohan”,”Shyam”,”Hari”,”Gopal”};

Name[0]	**R**	**A**	**M**	**\0**					
Name[1]	**M**	**O**	**H**	**A**	**N**	**\0**			
Name[2]	**S**	**H**	**Y**	**A**	**M**	**\0**			
Name[3]	**H**	**A**	**R**	**I**	**\0**				
Name[4]	**G**	**O**	**P**	**A**	**L**	**\0**			

POINTERS:

- A pointer is a variable that contains the address of a variable.
- Pointer is indicated by * symbol.
- *Is known as deferencing operator.

Pointer declaration and initialization:

Pointer declaration:

Declaration refers to the process of creating a pointer declaration of a pointer.

Syntax:

Data type * Ptr_ name;

Here data type is the data type of the value that the pointer will point to.

Example:

int * p; // p is a pointer to an integer

float *pch;

char *pfnum;

- Pointer variables can hold only address of other variables.
- The above three pointers are called dangling pointers because pointers are not specifically pointing to any particular variable

Example:

```
#include<stdio.h>

int main()

{

Printf("\n the size of short integer is :%d",sizeof(short int));

Printf("\n the size of unsigned integer is :%d",sizeof(unsigned int));

Printf("\n the size of signed integer is:%d",sizeof(signed int));

Printf("\n the size of integer is:%d",sizeof(int));

Printf("\n the size of long integer is :%d",sizeof(long int));
```

```
    Printf("\n the size of character is :%d",sizeof(char));
    Printf("\n the size of unsigned character is :%d",sizeof(unsigned char));
    Printf("\n the size of signjed character is :%d,size of(signed char));
    Printf("\n the size of floating point number is :%d",sizeof(float));
    Printf("\n the size of double number is :%d",sizeof(double));
    return 0;
}
```

Output:

The size of short integer is: 2

The size of unsigned integer is: 2

The size of signed integer is: 2

The size of integer is: 2

The size of long integer is: 4

The size of character is: 1

The size of unsigned character is: 1

The size of signed character is: 1

The size of floating point number is: 4

The size of double number is: 8

2

```
#include<stdio.h>
Int main()
{
     int num,*pnum;
     Pnum=&num;
```

```
    Printf("\n enter the number:");
    Scanf("%d",&num);
    Printf("\n the number that was entered is:%d",*pnum);
    Printf("\n the address of number in memory is :%p,&num);
    return 0;
}
```

Output:

Enter the number:10

The number that was entered is:10

The address of number in memory is:FFDC

POINTER FLEXIBILITY:

We can make the same pointer to point to different data variables in different statements.

Ex: int x, y, z, *p;

p = &x;

p = &y;

p = &z;

We can also use different pointers to point to the same data variable.

Ex: p1 = &x;

P2 = &x;

C P3 = &x

- The data type of the pointer variable and the variable whose address it will store must both be of the same type.

int x=10;

float y=2.0;

int *px;

int *py;

Px=&y; //invalid

Py=&x; //invalid

Initialization:

Process of assigning an address to the pointer.

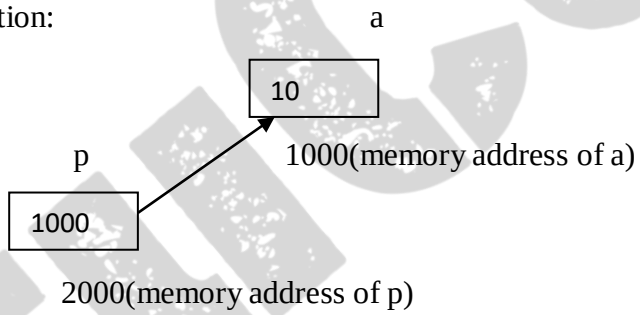
Example:

int a=10;

int *p;

p=&a;

memory representation:



Accessing variables through pointers:

- Accessing variable's values via pointers uses unary * operator.
- This is called as indirection or de-referencing operator.

Example:

int a=5;

int *p;

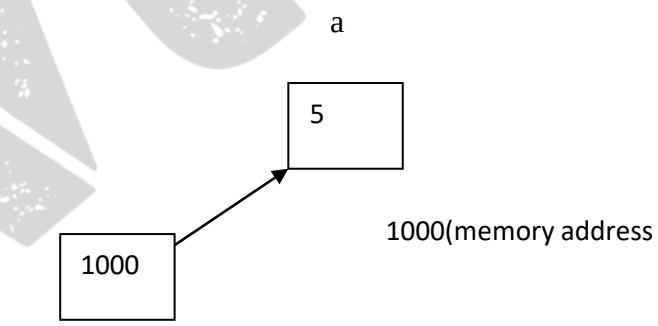
p=&a;

we access a value by *p

Example program:

Program to demonstrate pointers.

```
#include<stdio.h>
Void main()
{
int a=5 *p;    //declaration of variable a and pointer variable p
p=&a;          //initializing address of a to p
printf("a=%d\n",a);                    // 10
printf("&a=%d\n",&a);                  //1000
printf("p=%d\n",p);                    //1000
printf("*p=%d\n",*p);                  //10
printf("&p=%d\n",&p);                  //2000
}
```



2000(memorynaddress of p)

Types of pointers:

Typed pointer:

Pointer which has been declared with specific type of data type.

syntax:

data type *variable;

Example:

int *p;

float *p;

double *p;

Untyped pointer or generic pointer:

- A generic pointer is a pointer variable that has void as its data type.
- The void pointer or the generic pointer is a special type of pointer that can be used to point to variables of any data type.

Syntax:

Void * variable;

Example:

Void *ptr;

The void pointer will not point to any data and thus,cannot be dereferenced.You need to type cast a void pointer to another kind of pointer before using it.

Example:

```
#include<stdio.h>

int main()

{

    int x=10;
```

```
 Char ch='A';
Void *gp;
gp=&x;
Printf("\n generic pointer points to the integer value=%d",*(int*)gp);
gp=&ch;
printf("\n generic pointer now points to the character %c",*(char*)gp);
return 0;
}
```

Null pointer:

Null pointer which is a special pointer value that does not point anywhere. This means that a NULL pointer does not point to any valid memory address.

To declare a null pointer you may use the predefined constant NULL,which is defined in several standard header files including <stdio.h>,<stdlib.h> and <string.h>.

Syntax:

```
int *ptr =NULL;
```

you can always check whether a given pointer variable stores address of some variable or contains a NULL by writing,

```
if(ptr==NULL)
{
Statement block;
}
```

You may also initialize a pointer as a null pointer by using a constant 0 as follows:

```
int ptr;
ptr=0;
```

Passing arguments to functions using pointers:

- Call by value method of passing parameters to a function. Using call by value method, it is impossible to modify the actual parameters in call when you pass them to a function.

- The incoming arguments to a function are treated as local variable in the function and those local variables get a copy of the values passed from their caller.

The calling function sends the address of the variables and the called function must declare those incoming arguments as pointers.

In order to modify the variables sent by the caller,the called function must dereference the pointers that were passed to it.thus passing pointers to a function avoids the overhead of copying data from one function to another.

To pass pointers for passing arguments to a function the programmer must do the following:

1. Declare the function parameter as pointers.
2. Use the dereference pointers in the function body.
3. Pass the address as the actual argument when the function is called..

Example: Swapping of two numbers.

```
#include<stdio.h>
Void swap(int * n1,int *n2);
Void min()
{
        int a,b;
        printf("enter 2 numbers\n");
        scanf("%d%d",&a,&b);
        swap(&a,&b);
        printf("after swap a=%d\nb=%d",a,b);
}
Void swap (int *px,int *py)
{
         int hold;
        temp=*px;
        *px=*py;
```

```
    *py=temp;
}
```

Advantages of pointers:

- Pointers provide direct access to memory
- Pointers provide a way to return more than one value to the functions
- Reduces the storage space and complexity of the program
- Reduces the execution time of the program
- Provides an alternate way to access array elements
- Pointers can be used to pass information back and forth between the calling function and called function.
- A pointer allows us to perform dynamic memory allocation and deallocation.
- Pointers helps us to build complex data structures like linked list, stack, queues, trees, graphs etc.
- A pointer allows us to resize the dynamically allocated memory block.
- Addresses of objects can be extracted using pointers

Disadvantages of pointers:

- Uninitialized pointers might cause segmentation fault.
- Dynamically allocated block needs to be freed explicitly. Otherwise, it would lead to memory leak.
- Pointers are slower than normal variables.
- If pointers are updated with incorrect values, it might lead to memory corruption.