Character Operations in C'

- In C' programming language, total no. of characters are 256.
- When we are working with characters, all characters are represented with the help of an integer value, i.e. ASCII value of a character.
- The range of the ASCII value is from -128 to +127.
- Within the single quotation mark, any content is called 'character constant'.
- Always character const. returns an integer value, i.e. ASCII value of a character

Special Characters in C' :-

Character	Representation	ASCII value
'a'	a	97
'z'	z	122
'A'	A	65
'Z'	Z	90
'0'	0	48
'9'	9	57
'\n'	New line	10
'\t'	tab	9
'\\'	\	92
'\r'	Carriage return	13
'\b'	Backspace	8
'\"'	"	34
'\''	'	39
'%%'	%	37
'\%'	%	37
'\a'	Beep	7
'\0'	NULL	0
EOF	end of the line	-1
';'	;	59
':'	:	58
' '	Space	32

1112

1. printf("%d %d", 'a', 'A'); // 97 65
2. printf("%d %d", 'z', 'Z'); // 90 122
3. printf("%d %c", 'd', 100); // 100 d
4. printf("%c %c", 68, 99); // D c
5. printf("%d %d", '5', '9'); // 53 57
6. printf("%c %c", 'A'+32, 'a'-32); // a A

'A'+32 → 65+32 = 97 → 'a'

'a'-32 → 97-32 = 65 → 'A'

7. printf("welcome"); // welcome
8. printf("\"welcome\""); // "welcome"
9. printf("\'welcome\'"); // 'welcome'
10. printf("abc\nxyz"); // abc
xyz
11. printf("abc\\nxyz"); // abc\nxyz
12. printf("abc\\\nxyz"); abc\
xyz
13. printf("abc%dxyz", '\n'); abc10xyz
14. printf("abc%cxyz", 10); abc
xyz
15. printf("abc\txyz"); abc xyz

\t → 3 space (dos)
→ 5 space (output)

16. printf("abc%dxyz", '\t'); abc9xyz
17. printf("abc%cxyz", '\t'+1); abc
xyz
18. printf("Hello\b"); Hell
19. printf("Hello\b\babc"); // Helabc

Hel
abc

20. printf("Hello\b\b\b\b\babc"); abclo

Hello
abclo

21. printf("welcome\r"); welcome

22. printf(" welcome\r Hello"); Hellome

23. printf(" Hello\r Welcome"); Welcome

24. printf(" %%d ", 10); %d

25. printf(" %%d = %d", 10); %d = 10

26. printf(" abc\a "); abc beep (bell)

Buffer Concept :-

↳ Temporary storage area is called buffer.

↳ All standard i/o devices contain temporary buffer called standard i/o buffer.

↳ In implementation, when we are passing more than required no. of elements as a input, then automatically rest of elements will be stored in standard input buffer and these values will automatically pass to next input functionalities.

Ex)
```
#include <stdio.h>
#include<conio.h>
int main()
{
    int V1, V2;
    clrscr();
    printf("Enter value of V1 : ");
    scanf("%d", &V1);
    printf("\n Enter value of V2 : ");
    scanf("%d", &V2);
    printf("\n Sum of V1+V2 : %d", V1+V2);
    getch();
    return 0;
}
```

O/P
```
Enter value of V1 : 10
Enter value of V2 : 20
Sum of V1+V2 = 30
```

O/P
```
Enter value of V1 : 10 20 30 40
Enter value of V2 :
Sum of V1+V2 = 30
```

- In implementation, when we required to clear standard input buffer, then go for flushall() or fflush() function.

flushall()

↳ It is a predefined function, which is declared in standard input buffer <stdio.h>.

- By using this function, we can clear the standard input buffer data.

- Syntax: `void flushall(void);`

fflush() :-

- It is a predefined function, which is declared in <stdio.h>, by using this function, we can clear standard input buffer data.
- When we are working with fflush() function, it requires one argument of type FILE*, i.e. an address of a input buffer.
- (Syntax) `void fflush(FILE *stream);`

1)
```
#
#
int main()
{
    int v1, v2;
    clrscr();
    printf("\n Enter value of v1 : ");
    scanf("%d", &v1);
    printf("\n Enter value of v2 : ");
    flushall(); // fflush(stdin);
    scanf("%d", &v2);
    printf("\n Sum of %d + %d : %d", v1, v2, v1+v1);
    getch();
    return 0;
}
```

O/P 1
```
Enter value of v1 : 10
Enter value of v2 : 20
Sum of 10+20 : 30
```

O/P 2
```
Enter value of v1 : 10 20 30
Enter value of v2 : 20
Sum of 10+20 : 30
```

2)
```
#
#
int main()
{
    char ch1, ch2;
    clrscr();
    printf("\n Enter char1 : ");
    scanf("%c", &ch1);
    printf("\n Enter char2 : ");
    scanf("%c", &ch2);
    printf("\n ch1 = %c  ch2 = %c", ch1, ch2);
    getch();
    return 0;
}
```

1
```
Enter char1 : A (↵)
Enter char2 :
ch1 = A  ch2 =
```

2
```
Enter char1 : A B
Enter char2 :
ch1 = A  ch2 =
```

3
```
Enter char1 : AB
Enter char2 :
ch1 = A  ch2 = B
```

- When we are working with character operation, any keystroke on the keyboard is treated as characters only, that's why we are unable to read more than one characters properly.
- When we are working with scanf() function, it always reads the data from standard input buffer only, that's why we are unable to read more than one character by using scanf() function.
- Whenever we require to read more than one character, then always we require to use "flushall()" function

Ex)
```
#
#
int main()
{
    char ch1, ch2;
    clrscr();
    printf("\n Enter Char1 : ");
    // scanf("%c", &ch1);
    // ch1 = getchar();
    ch1 = getche();
    printf("\n Enter Char2 : ");
    // flushall();
    // scanf("%c", &ch2);
    ch2 = getch();
    printf("\n ch1 = %c  ch2 = %c", ch1, ch2);
    return 0;
}
```

O/P
```
Enter Char1 : A
Enter Char2 :
ch1 = A  ch2 = B
```

- When we are reading multiple characters, then always recommended to go for getche() or getch() function only.

getche()

- It's a predefined function, which is declared in "conio.h".
- By using this function, we can read a char directly from keyboard.
- When we are working with getche() function, then at a time it'll read single character from keyboard.
- When working with getche() funct°, then input character will be displayed on screen.

Syntax: `int getche(void);`

- "getche()" funct° returns an integer value, i.e. ASCII value of a character

getch()

- It's a predefined function, which is declared in "conio.h", by using this function, we can read a character directly from keyboard.
- when we are working with getch() funct^n, i/p characters ~~do not~~ is not displayed on console.
- getch() function returns an integer value, i.e ASCII value of a character.
- Syntax → `int getch(void);`

getchar(), putchar()

- getchar() is a predefined macro, which is defined in <stdio.h>.
- By using getchar(), we can read a character from stated input buffer.
- getchar() macro returns an integer value i.e. ASCII value of a character.
- Syntax → `int getchar(void);`

putchar()

- putchar() is a predefined macro which is defined in <stdio.h>.
- By using putchar() macro, we can print a character on console.
- putchar() macro requires one argument of type integer, i.e ASCII value of a character.

Syntax :- `int putchar(int ch);`

- When we are working with putchar() macro it returns an integer value, i.e. ASCII value of printed character.

Ex)
```
#include<stdio.h>
#include<conio.h>
int main()
{
    char ch1, ch2;
    printf("Enter a char : ");
    ch1 = getchar();
    ch2 = putchar();
    printf("\n char1 : %c  char2 : %c", ch1, ch2);
    getch();
    return 0;
}
```

O/P
```
Enter char1 : A
A
Char1 : A  char1 : A
```

(ex2)

```
#
#
int main()
{
    char ch1;
    int v1;
    clrscr();
    Pf("\n Enter a char (0-9): ");
    ch1 = getchar();
    if(ch1 >= '0' && ch1 <= '9')
    {
        v1 = ch1 - '0'; // v1 = '5' - '0'; 53 - 48 = 5
        Printf("\n Input value = %d", v1);
    }
    else
        printf("\n Invalid Input");
    getch();
    return 0;
}
```

```
Enter a char (0-9) : 5
Input value : 5
```

Strings :-

- Character array or group of characters or collection of characters are called String
- within the double quotation (" "), any content is called String content
- within the single quotation (' '), any content is called Character content
- Always string content will give base address of the string, character content will give ASCII value of character.
- When we are working with string content, always it should end with NULL character
- the representation of NULL character is '\0' and ASCII value is '0'.

Syntax: `char str[size];`

Properties of a String :-

① `char str[5];`

size → 5

sizeof(str) → 5B

② `char str[5];`

5 char variables

- str[0] → 1
- str[1] → 2
- str[2] → 3
- str[3] → 4

③ `char str[];` error

④ `char str[0];` error

⑤ `char str[-5];` error

- In declaration of the string, size must be unsigned type and their value is greater than zero.

(6) char str[5] = {a, b, c, d, e}; error

(7) char str[5] = {'a', 'b', 'c', 'd', 'e'};

a → str[0]
b → str[1]
c → str[2]
d → str[3]
e → str[4]

(8) char str[5] = {'A', 'B', 'C'};

str[0] = A
str[1] = B
str[2] = C
str[3] = NULL
str[4] = NULL

- On initialization of the string, if specific no. of characters are not initialized then rest of all data initialized with NULL.

(9) char str[2] = {'A', 'B', 'C'}; error

(10) char str[] = {'A', 'B', 'O', 'S'}; ✓

size → 4
sizeof(str) → 4B

- Generally, when we are initializing the string, always recommendent to initialize within the double quotes (" ") only.

(11) char str[10] = "welcome";

(12) char str[3] = "Hello"; error

(13) char str[] = {'H', 'E', 'L', 'L', 'O'};

size → 5
sizeof(str) → 5B

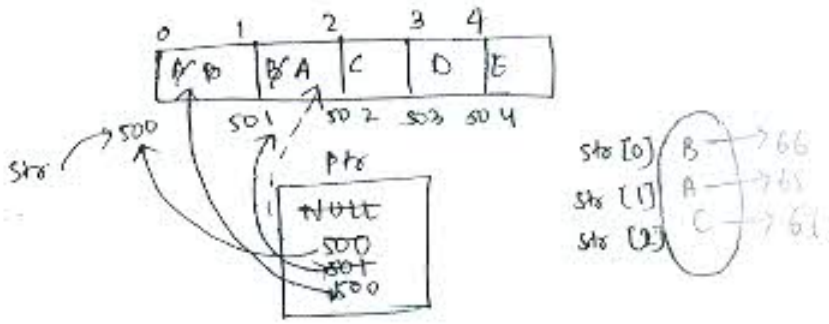
char str[] = "Hello";

size → 5
sizeof → 6B

- When we are working with a character array it doesnot required to end with NULL character, but if we are working with string contents, then it should end with NULL character only.

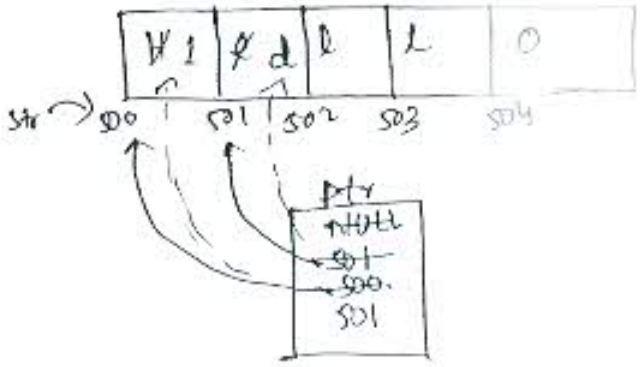
ex1) #
#
int main()
{
char Str[5] = { 'A', 'B', 'C', 'D', 'E' };
char near * ptr = (char near *) NULL;
ptr = &Str[0];
++ptr;
--*ptr;
--ptr;
++*ptr;
printf("%d %d %d", Str[0], Str[1], Str[2]);
getch();
return 0;
}

o/p:- 66 65 67



ex2) #
#
int main()
{
char Str[] = "Hello";
char * ptr = (char *) NULL;
ptr = Str + 1;
--ptr;
++*ptr; ++ptr; --*ptr;
printf("%c %c %c", Str[1], Str[0], Str[2]);
getch()
return 0;
}

o/p: dil



- String name always provides the base address of the string i.e. & Str[0].
- Str+1 will give next address of the string i.e. & Str[1]

(12)
```
#
#
int main()
{
  char str[10] = "Hello";
  printf("%c %c %c %c %c", str[0], str[1], str[2], str[3], str[4]);
  getch();
  return 0;
}
```

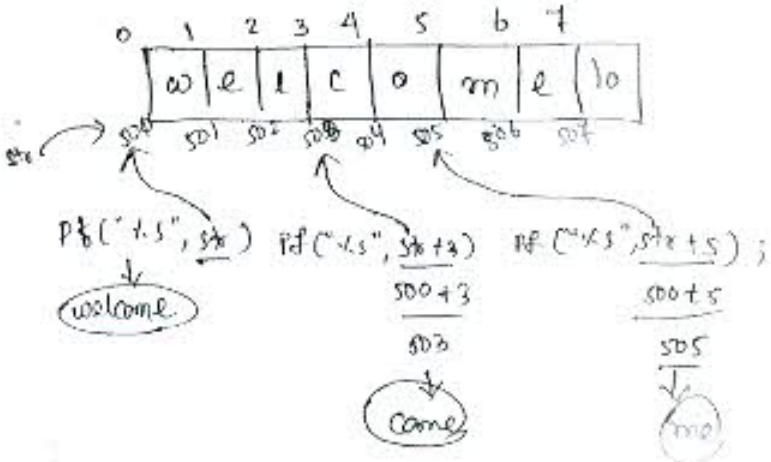
In implementation, when we are working with a string when we required to print entire content of the string in single format specifier then go for "%s" format specifier.

When we are working with "%s" format specifier, we required to pass an address of a string, from given address upto NULL, entire content will be printed on console.

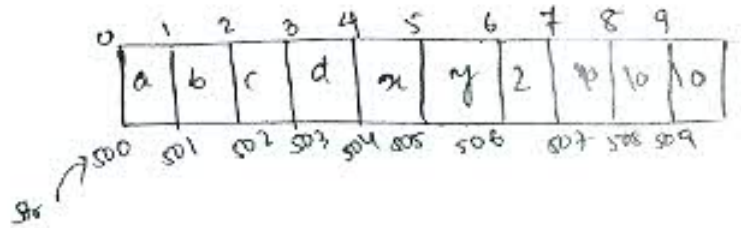
When we are working with characters type or character array, then recommended to go for '%c' format specifier

(13)
```
#include<stdio.h>
int main()
{
  char str[5] = "welcome";
  printf("%s", str);
}
```
O/P:- Error

(14)
```
#
#
int main()
{
  char str[] = "welcome";
  printf("\n %s", str);
  printf("\n %s", str+3);
  printf("\n %s", str+5);
  getch();
  return 0;
}
```
O/P
welcome
come
me



ex6)
```
#
#
int main()
{
    char str[6] = "abcdxyz";
    Printf("\n %s", str);
    Pf("\n %s", str+4);
    str[3] = 0; // str[3] = '\0';
    str[4] = 100; // str[4] = 'd'
    Printf("\n %s", str);
    Pf("\n %s", str+4);
    str[3] = 'D';
    str[5] = 0;
    Printf("\n %s", str);
    Pf("\n %s", str+6);
    getch();
    return 0;
}
```



o/p
```
abcdxyz
xyz
abc
dyz
abcD
z
```

ex7)
```
#
int main()
{
    char str[5] = {100, 65, 50, 95, 121};
    Pf("%s", str);
}
```

o/p: dA2_y

- even we are assigning numeric values in character array, then according to the ASCII values, corresponding characters will be stored

ex8)
```
#;
int main()
{
    char str[10] = "welcome";
    puts(str);
    Pf("%s", str);
    return 0;
}
```

o/p: welcome

puts()

- puts() is a predefined unformated function which is used to print the string content on console.
- when we are working with "puts()", it requires one argument of type const char * i.e. an address of a string.

syntax → int puts(const char *str);

- when we are working with "puts()" funct, automatically newline character will be printed after printing string content on console.

note → which function works with the help of format specifiers and can be applied for any datatype, is called formatted function.

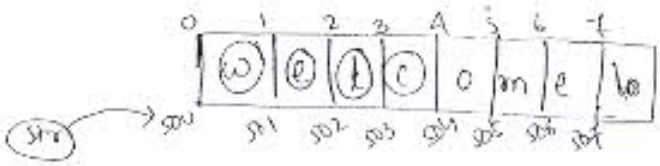
- ex:- printf(), scanf(), cprintf(), fprintf(), cscanf();

which function doesnot require format specifiers and need to be applied for specific datatypes only, is called unformated function.

ex:- puts(), gets(), getche(), cputs(), cgets()

```
#include <stdio.h>
#include <conio.h>
int main()
{
    char str[] = "welcome";
    int i;
    clrscr();
    for(i=0; str[i]!='\0'; i++)
        puts(str+i);
    getch();
    return 0;
}
```

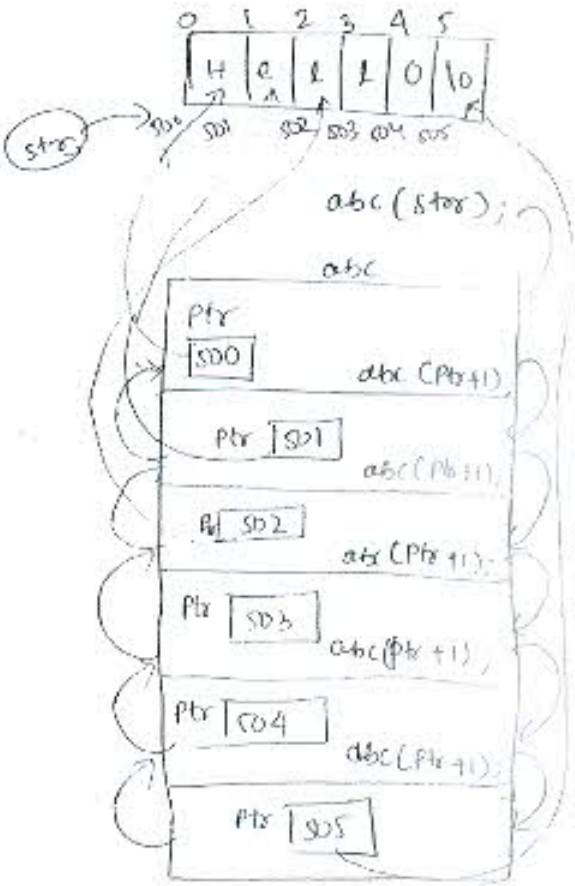
o/p:
welcome
elcome
lcome
come
ome
me
e



i	puts(str+i);	puts(str+i);	puts(str+i);	puts(str+i);
0	300+0	300+1	300+2	300+3
1	puts(300);	puts(301)	puts(302)	puts(303);
2	welcome	elcome	lcome	come
3				
4				
5				
6				
7				

ome me e

ex2)
```
# include <stdio.h>
# include <conio.h>
#
#
void abc (char * ptr)
{
    puts (ptr);
    if (*ptr)
        abc (ptr+1);
    puts (ptr);
}
void main ()
{
    char str[] = "Hello";
    abc (str);
    getch ();
}
```



op)

```
Hello
ello
llo
lo
o
-
-
o
lo
llo
ello
Hello
```

ex3)
```
#
#
int main ()
{
    char str[10] = "Rama Rao";
    puts (str);
    printf ("%s", str);
    getch ();
    return 0;
}
```

op)
```
Rama Rao
Rama Rao
```

ex1)
```
#
#
int main()
{
    char str[20];
    clrscr();
    printf("Enter a String : ");
    scanf("%s", str);
    printf("In the i/p string is : %s", str);
    getch();
    return 0;
}
```

o/p
Enter a String : Saumya Prakash
The i/p string is : Saumya

ex2)
```
#
#
int main()
{
    char str[20];
    clrscr();
    printf("In Enter a String : ");
    //scanf("%s", str);
    gets(str);
    printf("\n The i/p string is : %s", str);
    getch();
    return 0;
}
```

o/p (I)
Enter a string : welcome
The i/p string is : welcome

o/p (II)
Enter a String : Welcome Hello
The i/p string is : welcome Hello

- When we are working with the scanf() function, then space, tab & newline characters are treated as separators. That's why, when the separator is occurred, it is replaced with null character.
- When we are working with the scanf() function, we can read the string data, if it does not have any separators.
- In implementation, when we are working with the strings, always it's recommended to go for "gets()" function only

gets()
- It's a predefined unformatted function, which is declared in "stdio.h".
- By using this predefined function, we can read the string data properly.
- When we are working with the gets() function, new line character only is treated as separator.
- Syntax:

```
char * gets(char *str);
```

ex:5)
```
#
#
int main()
{
    char s1[10] = "Hello";
    char s2[10] = "welcome";
    clrscr();
    puts(s1);
    puts(s2);
    s2=s1; // 500 = 600; error
    puts(s1);
    puts(s2);
    getch();
    return 0;
}
```

(o/p) ~~Hello welcome~~ error: L value required.

- No any string operations are possible directly by using operators.
- In implementation, when we require the string manipulations, then recommended to go for any predefined string library functions or implement user defined function.
- All string related predefined functions are available in "string.h"; those are :-

① strcpy();
② strlen();
③ strrev();
④ strcat();
⑤ strupr();
⑥ strlwr();
⑦ strcmp();
⑧ stricmp();

① strcpy()

- By using this function, we can copy a string data to another string.
- When we are working with strcpy() function, it requires two arguments of type char *; i.e. an address of a string.
- Syntax: `char * strcpy(char * dest, const char * src);`
- When we are working with strcpy() function from given source address upto null char content will be copied to destination string.

Ex>

```
#
#
#include<string.h>
int main()
{
  char s1[10] = "welcome";
  char s2[10] = "Hello";
  clrscr();
  puts(s1);
  puts(s2);
  strcpy(s1, s2); // s1 = s2
  puts(s1);
  puts(s2);
  getch();
  return 0;
}
```

Op:

welcome
Hello
Hello
Hello

1. strcpy(s1, s2);

welcome
Hello
Hello
Hello

2. strcpy(s1, s2+2);

welcome
Hello
llo
Hello

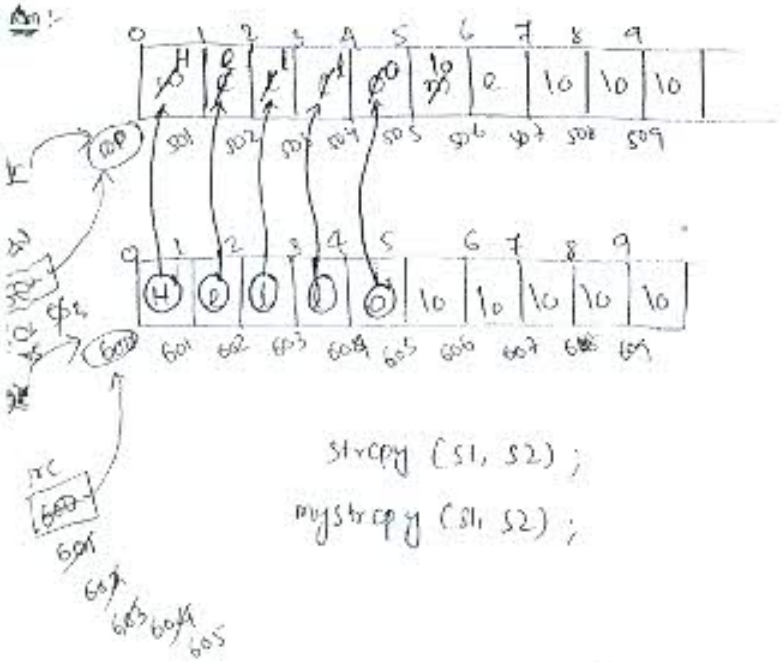
3. strcpy(s1+3, s2);

welcome
Hello
welHello
Hello

4. strcpy(s1+3, s2+2);

welcome
Hello
Wello
Hello

7> copy one string to another string without using "strcpy()" function




```
void mystrcpy(char *dest, const char *src)
{
  while(*src != '\0')
  {
    *dest = *src;
    src++;
    dest++;
  }
  *dest = '\0';
}
```

Ans :-

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void mystrcpy (char *dest, const char *src)
{
    while (*src != '\0')
    {
        *dest = *src;
        dest++;
        src++;
    }
    *dest = '\0';
}
int main()
{
    char s1[10] = "welcome";
    char s2[10] = "Hello";
    clrscr();
    puts(s1);
    puts(s2);
    mystrcpy(s1, s2); // strcpy(s1, s2);
    puts(s1);
    puts(s2);
    getch();
    return 0;
}
```

o/p

```
Welcome
Hello
Hello
Hello
```

(2) strlen()

- By using this function, we can find the length of a string.
- Length of the string means, total no of characters excluding '\0' character (null character).
- When working with "strlen()" function, it requires one argument of type 'const char *' i.e. an address of a string, and it returns an 'int' value.

Syntax:

```
int strlen(const char *str);
```

- When working with 'strlen()' function from given address upto null terminal no of character count value will be passed back.

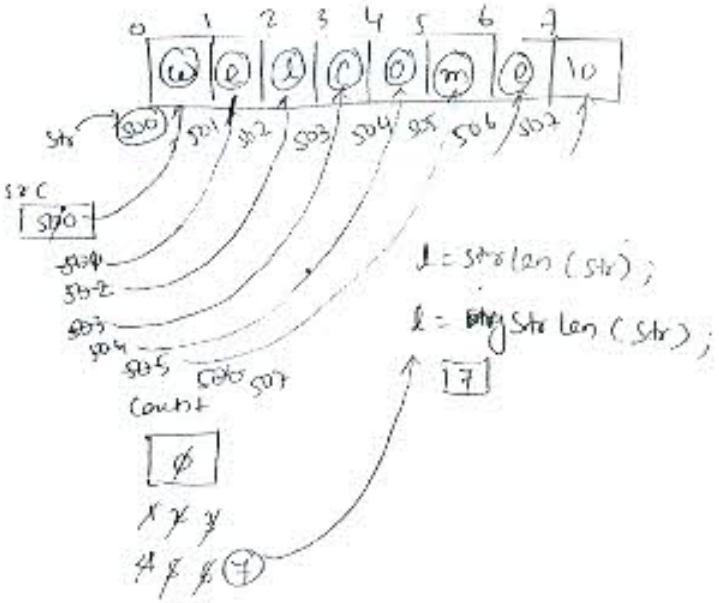
x8) #
#
#
int main ()
{
char str[] = "welcome";
int l, s;
clrscr();
puts(str);
s = sizeof(str);
l = strlen(str);
printf("\n length of string : %d", l);
printf("\n size of string : %d", s);
getch();
return 0;
}

o/p
welcome
length of string : 7
size of string : 8

① strlen(str); // 7
② strlen(str+3); // 4
③ strlen(str+5); // 5
④ strlen(str+7); // 7

- using user defined function →

⇒ int myStrlen(const char *src)
{
int count = 0;
while (*src != '\0')
{
++count;
src++;
}
return count;
}



9/11/12

eg>
```
#
#include <string.h>
int mystrlen (const char *src)
{
   int count = 0;
   while (*src != '\0')
   {
      count++;
      ++src;
   }
   count++; return count;
   ++src;
}
int main ()
{
  char str[10];
  int s, l;
  clrscr();
  printf(" Enter a string :");
  gets (str);
  l = mystrlen (str); // l = strlen(str);
  s = sizeof (str);
  Printf (" length of str = %d ", l);
  Printf (" Size of str = %d", s);
}
```

o/p

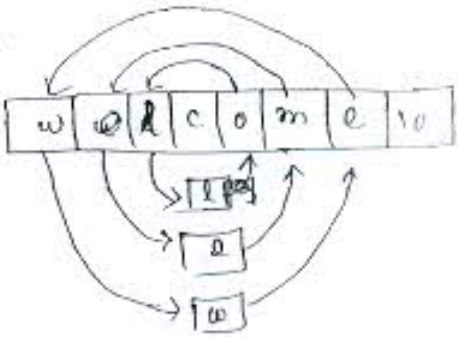
length = 7
size = 10

(3) Strrev () :-

- By using strrev (), we can make the string as reverse.
- strrev () requires one argument of type char *, i.e. an address of a string.
- When we are working with strrev() from given address upto null, entire content will be made Reverse (excluding '\0' null char).

char * strrev (char * str);

```
#
#
#include <string.h>
int main()
{
   char str[10] = "welcome";
    clrscr();
    puts(str);
    strrev(str);
    printf("In string in Reverse is : %s", str);
    getch();
}
```



① welcome → emoclew

② strrev(str+2) → welcome

③ strrev(str+5) → welcom

④ strrev(str+7) → welcome

(12)
```
#
#
void mystrrev(char *src)
{
    int i, l;
    char t;
    l = strlen(src);
    for(i=0; i<l/2; i++)
    {
        t = src[i];
        src[i] = src[l-i-1];
        src[l-i-1] = t;
    }
}
int main()
{
    char str[20]
    clrscr();
    printf("Enter a string:");
    gets(str);
```

```
    mystrrev(str); // strrev(str)
     printf("Reverse string data : %s", str);
     getch();
}
```

(4) strcat() :-

- By using strcat(), we can append a string to another string.
- strcat() requires two arguments of type char *, that is an address of a str
- When we are working with strcat(), always appending will happen at end of the destination string only.

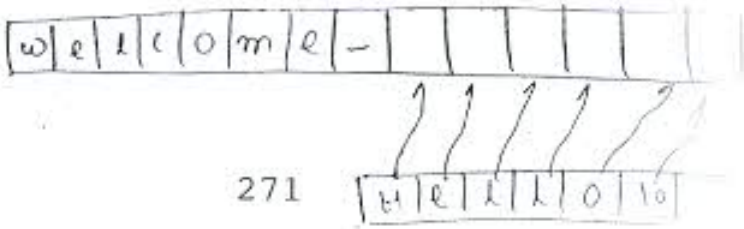
Syntax: char *strcat(char *dest, const char *sr()

Ex:

```
#
#
#
int main()
{
    char s1[20] = "welcome";
    char s2[10] = "Hello";
     clrscr();
     puts(s1);
     puts(s2);
     strcat(s1, " ");
     strcat(s1, s2);
      puts(s1); puts(s2);
      getch();
}
```

welcome
Hello
welcome Hello

→ strcat(s1, s2) welcome Hello
→ strcat(s1, s2+2) welcome Hello
→ strcat(s1+3, s2) welcome Hello
→ strcat(s1+3, s2+2) welcome llo



(x2) 11

```
#include
#include
void mystrcat(char *dest, const char *src)
{
    int l;
    l = strlen(dest);
    dest = dest + l;
    while(*dest++ = *src++)
    {
    }
    while(*src != '\0')
    {
        *dest = *src;
        dest++;
        src++;
    }
}
int main()
{
    char s1[20] = "welcome";
    char s2[10] = "Hello";
    clrscr();
    puts(s1);
    puts(s2);
    mystrcat(s1, " "); // strcat(s1, " ");
    mystrcat(s1, s2); // strcat
    puts(s1);
    puts(s2);
    getch();
    return 0;
}
```

⑤ strupr() :-

- By using this function, we can convert the string into upper case.
- strupr() function requires one argument of type char *, that is an address of a string.
- When we are working with strupr() from given address, upto null, entire lower case content will be constructed converted into upper case.

```
char * strupr (char *str);
```

ex) #
#
#
int main()
{
char str[10];
clrscr();
printf("Enter a string:");
gets(str);
strupr(str);
printf("In upper case String: %s", str);
getch();
}

welcome ⟶ WELCOME

strupr(str+3) → welCOME
strupr(str+5) → welcoME
strupr(str+7) → welcome

- When we are working with strupr() from given address upto null, char by char comparisons will take place for lower case content. If lower case content is occurred, then it is converted into upper case.

ex) #
#
#
void mystrupr(char *str)
{
int i;
for(i=0; str[i]!='\0'; i++)
{
if(str[i]>='a' && str[i]<='z')
str[i]=str[i]-32;
}

```
int main()
{
    char str[10];
    clrscr();
    printf("Enter a string:");
    gets(str);
    mystrupr(str); // strupr(str)
    printf("In upper case string:%s", str);
    getch();
}
```

⑥ strlwr() :-

- By using this function, we can convert a string into lower case.
- strlwr() func^{n} requires an argument of type char *, i.e. an address of a string.
- When we are working with strlwr() from given address, all upper case content will be converted into lower case.

```
char * strlwr(char *str);
```

Ex) #

```
#
#
void mystrlwr(char *str)
{
  while(*str != '\0')
  {
    if(*str >= 'A' && *str <= 'Z')
        *str = *str + 32;
    ++*str;
  }
}
int main()
{
    char str[10];
    clrscr();
    printf("Enter a string:");
    gets(str);
    mystrlwr(str); // strlwr(str);
    printf("Lower case string:%s", str);
    getch();
}
```

strcmp()

- By using this predefined function, we can compare two strings.
- strcmp() function requires two arguments of type const char * and returns an integer value.
- Syntax: int strcmp (const char * s1, const char * s2);
- When we are working with strcmp function, characters by character comparison will take place until first unpaired character is occurred.
- When we are working with strcmp() function, then first unpair character is occurred then it returns ASCII value difference.
- If both strings contain same data, then it returns '0'

Ex1>
```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main()
{
    char s1[10] = "welcome";
    char s2[10] = "welcome"; int d;
    clrscr();
    puts(s1);
    puts(s2);
    d = strcmp(s1, s2);
    printf("ASCII value diff : %d", d);
    getch();
    return 0;
}
```

strcmp(s1, s2);
myStrcmp(s1, s2);

Ex2>
```
/* using userdefined function */
int mystr (const char *s1, const char *s2)
{
    int d=0;
    while (*s1 != '\0' || *s2 != '\0')
    {
        if (*s1 != *s2)
        {
            d = *s1 - *s2;
            return d;
        }
        s1++;
        s2++;
    }
    return 0;
}
```

#2) #
#
#

```
void main()
int mystrcmp(const char *s1, const char *s2)
{
    int d;
    while(*s1!='\0' || *s2!='\0')
    {
        if(*s1!=*s2)
        {
            d = *s1 - *s2;
            return d;
        }
        s1++;
        s2++;
    }
    return d;
}
int main()
{
    char *s1[10]= "welcome";
    char s2[10]="welcome";
    int d;
    clrscr();
    puts(s1);
    puts(s2);
    d = mystrcmp(s1, s2); // d = strcmp(s1, s2);
    printf("ASCII value diff : %d", d);
    getch();
    return 0;
}
```

- When we are working with strcmp() function, it works with the help of case-sensity, i.e. uppercase & lowercase content both are different.
- On implementation when we need to ignore the case-sensitivity, then go for "strcmp()" function.

stricmp()

- By using this function, we can compare two strings without any case, i.e. uppercase & lowercase content both are treated as same.
- stricmp() function requires two arguments of type "const char *" i.e. address of two strings and returns an integer value.
- when we are working with stricmp() function, then character by character comparison will take place until first unpaired character is occurred, when first unpaired character is occurred, then returns ASCII value difference.

- Syntax ⇒ `int stricmp (const char * s1 , const char * s2 );`

Ex1)
```
#
#
#
int main ()
{
    char s1[10] = "ABC";
    char s2[10] = "abc";
    int d;
    clrscr();
    puts(s1);
    puts(s2);
    d = stricmp (s1, s2);
    printf("\n ASCII value diff. is : %d", d);
   getch();
   return 0;
}
```

o/p

```
ABC
abc
ASCII value diff. is : 0
```

(I)
s1 → ABC } 0
s2 → ABC

s1 → abc } 0
s2 → abc

s1 → ABC } -32
s2 → abc

s1 → abc } +32
s2 → ABC

} 0

(II)
s1 → BCA } (B)
s2 → @bc } (@)
(B) } (+1)
a - 32 = (A)

(III)
s1 → abc } (A) (-32)
s2 → BCA } (B)
} (-1)

(IV)
s1 → a b c } a
s2 → b c a } b
} (-1)

(V)
s1 → A B C } A
s2 → 2^{3}ZZ A } B
} (-1)

(VI)
s1 → abc } a (32)
s2 → null } null
} (65)

(VII)
s1 → NULL } NULL
s2 → abc } a
} (-97)

(2)

```
#
#
#
int mystricmp (const char * s1, const char * s2)
{
    int d1, d2, t1, t2;
    while (*s1 != '\0' || *s2 != '\0')
    {
        if (*s1 - *s2 == 32 || *s1 - *s2 == -32 || *s1 - *s2 == 0)
        {
            s1++; // case 1;
            s2++;
        }
        else
        {
            t1 = *s1;
            t2 = *s2;
            if (t1 >= 'a' && t1 <= 'z' || t2 >= 'a' && t2 <= 'z')
            {
                if (t1 >= 'a' && t1 <= 'z' && !(t2 >= 'a' && t2 <= 'z'))
                {
                    t1 -= 32; // case 3, case 6
                    return (t1 - t2);
                }
                else
                if (t2 >= 'a' && t2 <= 'z' && !(t1 >= 'a' && t1 <= 'z') && t1 != 0)
                {
                    t2 -= 32; // case 2
                    return (t1 - t2);
                }
                else
                    return (t1 - t2); // case 4, case 7
            }
            else
                return (t1 - t2); // case 5
        }
    }
    return 0;
}
```

```
int main ()
{
    char s1[10]; char s2[10];
    int d1, d2;
    Printf(" Ents str1 :");
    gets(s1);
    Printf(" Ents str2 :");
    gets(s2);

    d1 = Stricmp (s1, s2);
    d2 = mystricmp(s1, s2);
    printf("%d %d", s1, s2);
    getch();
    return 0;
}
```

Ex3> Enter a string: welcome AS9#*:? 30500978017T
Total count : 15

```
#
#
int alphacount (char str[])
{
    int count=0, i;
    for (i=0 ; str[i] != '\0'; i++)
    {
        if (str[i] >='A' && str[i] <='Z' || str[i] >='a' && str[i] <='z')
            ++count ;
    }
    return count;
}
int main()
{
    char str[50];
    int c;
    clrscr();
    printf(" Enter a string : ");
    gets(str);
    c= alphacount(str);
    printf("\n Total count : %d", c);
    getch();
    return 0;
}
```

Ex4> vowel program

```
#
#
int main()
{
    char str[50];
    int i;
    clrscr();
    printf("\n enter a string : ");
    gets(str);
```

```
for (i=0; str[i] != '\0'; i++)
{
  switch (str[i])
  {
    case 'A':
    case 'a': ++count;
              break;
    case 'E':
    case 'e': ++count;
              break;
    case 'I':
    case 'i': ++count;
              break;
    case 'O':
    case 'o': ++count;
              break;
    case 'U':
    case 'u': ++count;
              break;
  }
}
printf("\n Total Count = %d", count);
getch();
return 0;
}
```

```
switch (str[i])
{
    case 'A':
    case 'a':
    case 'E':
    case 'e':
    case 'I':
    case 'i':
    case 'O':
    case 'o':
    case 'U':
    case 'u': ++count;
}
```

O/P

```
Enter a String: Soumya
Total Count : 4
```

5) / palindrome program */

```
#
#
#include <string.h>
int main()
{
   char str[10];
   char temp[10];
   clrscr();
   printf("Enter a string : ");
   gets(str);
   strcpy(temp, str);
   strrev(temp);
   if (strcmp(str, temp) == 0)
        printf("\n %s is palindrome", str);
```

```
    else
      printf ("\n %s is not a palindrome", str);
    getch();
    return 0;
}
```

o/p

```
Enter a string : MaDaM
MaDaM is a palindrome
```

Ex6) Enter a String : 91 98 24217 98 55598 9848418 75421 20100

```
0 : * * *
1 : * * * * *
2 : * * * *
4 : * * * *
. . . .
. . . . .
```

```
#
#
int main()
{
   char str[50];
   char ch;
   int i, count = 0;
   clrscr();
   printf("\n Enter a String : ");
   gets(str);
   // strupr(str);
   /* for alphabet count */
   // for (ch = 'A'; ch <= 'Z'; ch++)
   for (ch = '0'; ch <= '9'; ch++)
   {
      count = 0;
      for (i = 0; str[i] != '\0'; i++)
      {
         if (ch == str[i])
            ++count;
      }
      if (count > 0)
      {
         printf("\n %c : ", ch);
```

```
        while (count > 0)
        {
            printf ("*");
            --count ;
        }
    }
}
getch ();
return 0;
}
```

ex) #

#

```
int main ()
{
  char str [4] [20] = {
                        "Oracle",
                        "Java",
                        "COBOL",
                        "PASCAL"
                      };
  char * ptr [4];
  char ** pptr ;
  int i;
  clrscr();
  ptr [0] = str + 0 ; // str [0] ; // & str [0] [0] ;
  ptr [1] = str + 1 ; // str [1] ; // & str [1] [0] ;
  ptr [2] = str + 2 ; // str [2] ; // & str [2] [0] ;
  ptr [3] = str + 3 ; // str [3] ; // & str [3] [0] ;
  pptr = ptr ; // pptr = & ptr [0] ;
  for (i = 1 ; i <= 4 ; i++)
  {
     * pptr += i ; // * pptr = * pptr + i ;
     ** pptr += i ; // ** pptr = ** pptr + i ;
     ++ pptr ;
  }
```

Interview Questions

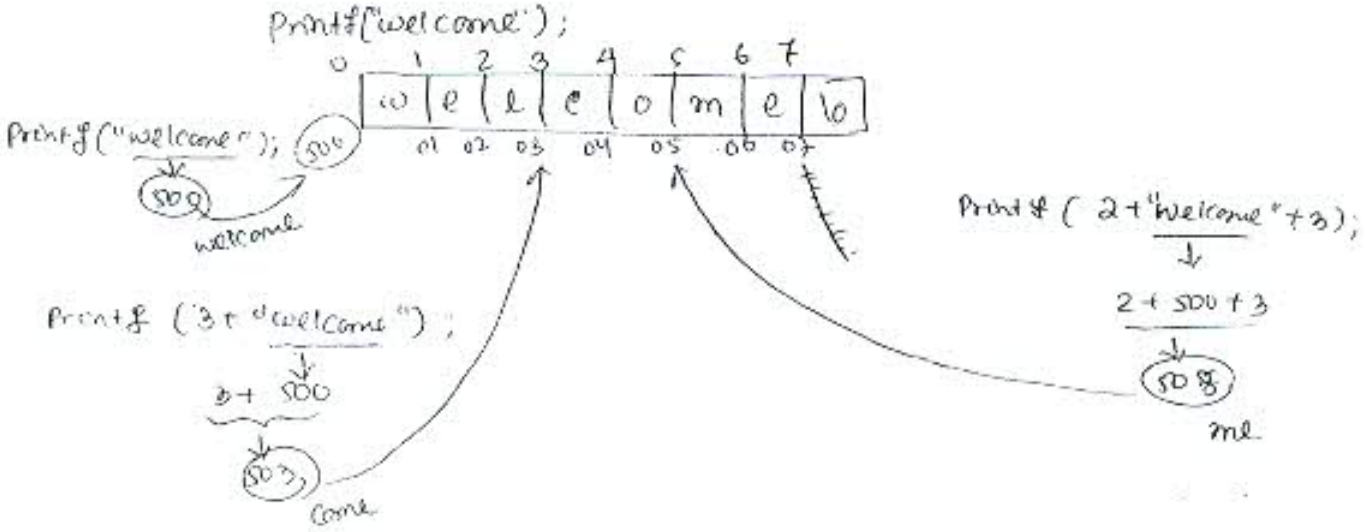
(Q1)
```
#
int main()
{
  printf("welcome");
  return 0;
}
```

(O/P) welcome

① printf("welcome"); → welcome

② printf("3+"welcome"); → come

③ printf(3+"welcome"+3); → me



(Q2)
```
#
int main()
{
  puts("welcome");
  return 0;
}
```

(O/P) welcome

① puts("welcome"); → welcome

② puts(3+"welcome"); → come

③ puts(3+"welcome"+3); → me

(Q3)
```
#
int main()
{
  char str[10] = "welcome";
  printf(str);
  return 0;
}
```

(O/P) welcome

① printf(str); → welcome

② printf(3+str); → come

③ printf(2+str+3); → me

Q4) #
```
int main()
{
    printf("welcome" "Hello");
    return 0;
}
```
1. printf("welcome", "Hello"); → welcomeHello
2. printf("welcome%s", "Hello"); → welcomeHello

Ans) welcomeHello

Q5) #
```
int main()
{
    char str[20] = "%d Naresh IT %d";
    printf(str);
    return 0;
}
```
1. printf(str); → garbage Naresh IT garbage
2. printf(str, 10, 20) → 10 Naresh IT 20

Ans) garbage Naresh IT garbage

Q6) #
```
int main()
{
    char str[20] = "%d Hello %d";
    puts(str);
    return 0;
}
```
1. puts(str); → %d Hello %d
2. puts(str, 10, 20); → error

Q7) #
```
int main()
{
    printf("%s\n", "welcome");   // welcome
    printf("%.3s\n", "welcome"); // wel
    printf("%.3s\n", 2+"welcome"); // lco
    return 0;
}
```

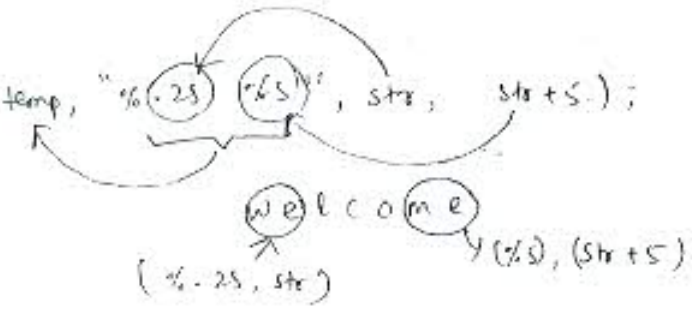
Q8) #
```
int main()
{
    char str[10] = "welcome";
    printf("%s\n", str);   // welcome
    printf("%.3s\n", str); // wel
    printf("%.3s\n", 2+str); // lco
    return 0;
}
```

Q9) B

```
int main()
{
    char str[10] = "welcome";
    char temp[10];
    printf("%s\n", str); //welcome
    printf("%.3s\n", str); //wel
    printf("%.*s\n", 2+str); //lco
    sprintf(temp, "%.2s %s", str, str+5);
            //nothing prints, but copy the string onto temp;
    printf("\n%s", temp);
    return 0;
}
```

note)

sprintf(temp, "%.2s %s", str, str+5);



temp → welme

printf("%s", temp); //welme

DYNAMIC MEMORY - ALLOCATION

- When we are working with or allocating the memory at runtime is called DMA.
- By using DMA, we can allocate or deallocate the memory dynamically, that's at the time of execution of the program.
- By using DMA, wherever we want, which type we want & how much we want that time, that type & that much we can construct dynamically.
- In C programming language, 2 types of memory management are available, such as :-

  - static memory management (Compile time memory mgmt)
  - Dynamic memory mgmt (Runtime memory mgmt)

- When we are creating the memory at the memory at the time of compilation, then it's called compile time memory mgmt or static memory mgmt.
- When we are working with static memory mgmt, we can't extend memory, if it's not sufficient.
- When we are working with static memory mgmt, we can't manage / utilize the memory properly.
- ex:- variable declarations, arrays, strings.
- To overcome the static memory mgmt problem, we require to go for dynamic memory allocation.
- By using DMA, we can utilize the memory more efficiently.
- DMA related all predefined functions are available in following header files.

  i) <alloc.h>
  ii) <malloc.h>
  iii) <mem.h>

- DMA related predefined functions are :-

  1. malloc()
  2. calloc()
  3. realloc()
  4. free()
  5. farmalloc()
  6. farcalloc()
  7. farrealloc()
  8. farfree()

① malloc()

- By using malloc() function, we can create the memory dynamically at initial stage
- malloc() function requires one argument of type size_type, i.e datatype size
- malloc() function creates the memory in bytes format.
- malloc() through created memory, initial value is garbage.

Syntax: `void * malloc(size_type);`

Note: When we are working with DMA related functions, then it can be applied for any datatype, that's why DMA related functions return void *, i.e generic type.

- When we are working with DMA related functions, then we require to use type-casting process, because function returns void *.

```
int *ptr;
ptr = (int *) malloc(sizeof(int)); // 2B

float *ptr;
ptr = (float *) malloc(sizeof(float)); // 4B

char *ptr;
ptr = (char *) malloc(sizeof(char)); // 1B

int *arr;
arr = (int *) malloc(sizeof(int) * 10); // 20B

char *str;
str = (char *) malloc(sizeof(char) * 50); // 50B
```

② calloc() :-

- By using calloc() function, we can create the memory dynamically at initial stage
- calloc() function requires two arguments of type count, size_type
- count is an integer variable, which indicates no. of elements.
- size_type is datatype size, which indicates actual size of a variable
- When we are working with calloc() function, it creates the memory in stack format & initial value is zero.
- Syntax: `void * calloc(count, size_type);`

ex>
```
int *arr;
arr = (int *) calloc (10, sizeof(int)) ; // 20B

char *str;
str = (char *) calloc (9, sizeof (char)) ; // 9B
```

③ realloc() :-

note:- when we need to create the memory for a variable, then go for malloc(); if we are creating the memory for an array, then recommended to go for calloc().

- By using this predefined function (realloc()), we can create the memory dynamically at middle stage or ending stage of the program.
- realloc() function requires two arguments of type void *, size_type.
- void * indicates previous block base address, size_type is datatype size.
- When we are working with realloc() function, it creates the memory in byte format & the initial value is garbage.
- Syntax:- `void * realloc( void *, size_type );`

ex>
```
int *arr;
arr = (int *) calloc (5, sizeof(int)) ; // 10 B
.....
.....
.....
arr = (int *) realloc (arr, sizeof(int) * 10) ; // 20B
```

④ free() :-

- DMA related memory are always created in heap area of Data segment
- DMA related memory is permanent memory, if it is not deallocated, that's why always it is recommended to deallocate the memory at end of the program.
- By using free() function, we can deallocate dynamically created memory.
- free() function requires one argument of type void *, i.e. an address of a memory.
- Syntax:- `void free (void*);`

- ex>
```
int *arr;
arr=(int*) calloc(10, sizeof(int));
.....
.....
.....
free(arr);
```

note:- By using malloc(), calloc(), realloc() functions, we can create maximum of 64 KB data only; when we need to create more than 64 KB data recommended to go for far malloc(), far calloc() & far realloc() functions.

- By using free() function, we can deallocate 64 Kb data only, when we need to deallocate more than 64 Kb data, then recommended to go for farfree() function

5) farmalloc() :-

```
void far *farmalloc (size_type) ;
```

ex)
```
int far * arr ;
arr = (int far *) farmalloc (sizeof(int) * 5000) ;
```

6) farcalloc() :-

```
void far* farcalloc (uint, size_type) ;
```

ex)
```
int far * arr ;
arr = (int far *) farcalloc (1000, sizeof (long int)) ;
```

7) farrealloc() :-

```
void far * farrealloc ( void far *, size_type ) ;
```

ex)
```
int far * arr ;
arr = (int far *) farralloc ( 5000, sizeof (long int)) ;
. . . .
. . . .
arr = (int far *) farrelloc ( arr, sizeof (long) * 5000) ;
```

8) farfree() :-

```
void farfree( void far *) ;
```

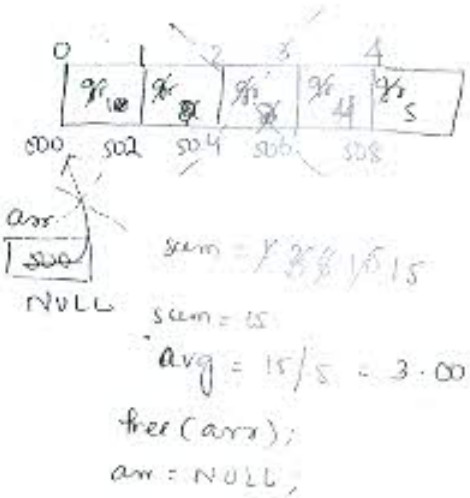
ex)
```
int far * arr ;
arr = (int far *) farcalloc ( 500, sizeof (long int)) ;
. . . .
. . . .
farfree (arr) ;
```

ex)
```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
int main()
{
    // int arr[10] ;
    int * arr ;
    int size, sum=0, i ;
    float avg ;
    clrscr() ;
```

```
Printf("\n Enter array size : ");
scanf("%d", &size);
arr = (int *) malloc (sizeof(int) * size);
Printf("\n Default values : ");
for (i=0; i<size; i++)
    Printf(" %d ", arr[i]);
Printf(" Enter %d values : ", size);
for(i=0; i<size; i++)
{   scanf("%d", &arr[i]
    sum += arr[i];
}
avg = (float)Sum / size;
printf("\n Sum = %d", Sum);
Printf("\n Average of Sum = %f", avg);
free(arr); arr = NULL;
getch();
return 0;
}
```

OR

```
Enter array size 5
Default values : 12345 -3457 12345 3927 &2468
Enter 5 values : 1 2 3 4 5
Sum = 15
Average of Sum = 3.00
```

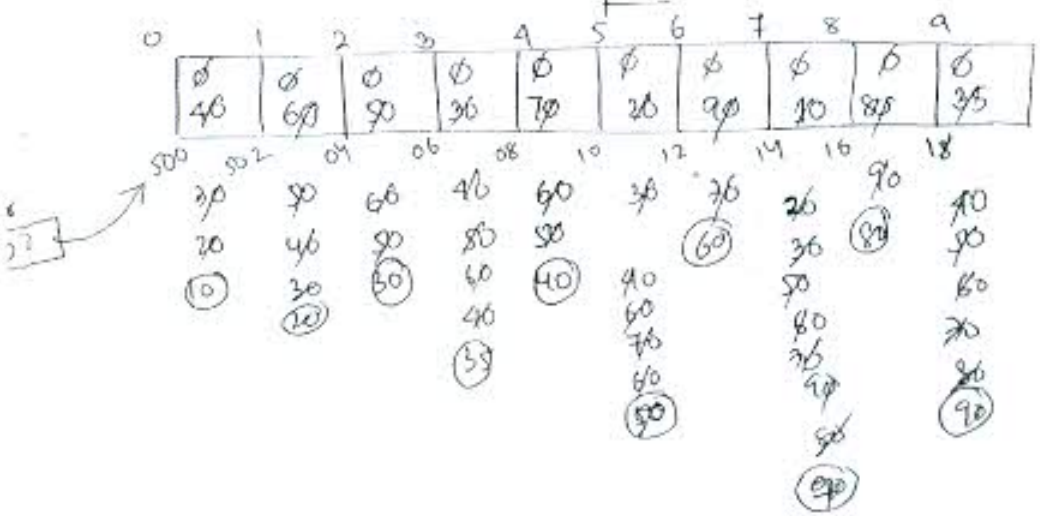


Ex 2)

```
#
#
#include <alloc.h>
int main()
{
    int * arr;
    int size, i, j, t;
    clrscr();
    printf("\n Enter Array Size : ");
    scanf(" %d", &size);
    arr = (int *) calloc (size, sizeof(int));
    printf("\n Default values are :");
    for(i=0; i<size; i++)
        printf("%d ", arr[i]);
```

```
Printf("Enter %d values : ", size);
for (i=0; i<size; i++)
        scanf("%d", &arr[i]);
//selection sort logic
for (i=0; i<size; i++)
{
  for(j=i+1; j<size; j++)
  {
      if (arr[j] < arr[i])
      {
            t = arr[i];
            arr[i] = arr[j];
            arr[j] = t;
      }
  }
}
printf("\n Sorted Arr. data: ");
for (i=0; i<size; i++)
      printf("%d ", arr[i]);
getch();
free(arr);
arr = NULL;
return 0;
}
```

```
Enter Array Size : 10
Default values are : 0 0 0 0 0 0 0 0 0 0
Enter 10 values : 40 60 50 30 70 20 90 10 80 35
Sorted Arr. data : 10 20 30 35 40 50 60 70 80 90
```

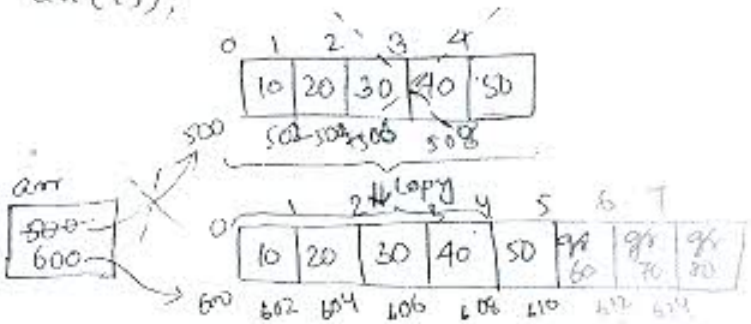


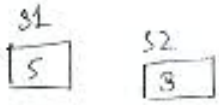
20.11.22

realloc() :-

ex:)

```
#
#
#include <alloc.h>
int main()
{
   int *arr;
   int s1, s2, i;
   clrscr();
   printf(" Enter size1 value : ");
   scanf("%d", &s1);
   arr = (int *) calloc (s1, sizeof (int));
   printf(" In Enter %d values :", s1);
   for(i=0; i<s1; i++)
         scanf("%d", &arr[i]);
   printf("In Enter size2 value : ");
   scanf("%d", &s2);
   arr = (int *) realloc (arr, sizeof(int) * (s1+s2));
   printf(" In Enter %d values : ", s2);
   for(i=s1; i<s1+s2; i++)
         scanf("%d", &arr[i]);
   printf("In Arr List is : ");
   for(i=0; i<s1+s2; i++)
         printf("%d", arr[i]);
   getch();
   free(arr);
   arr = NULL;
   return 0;
}
```





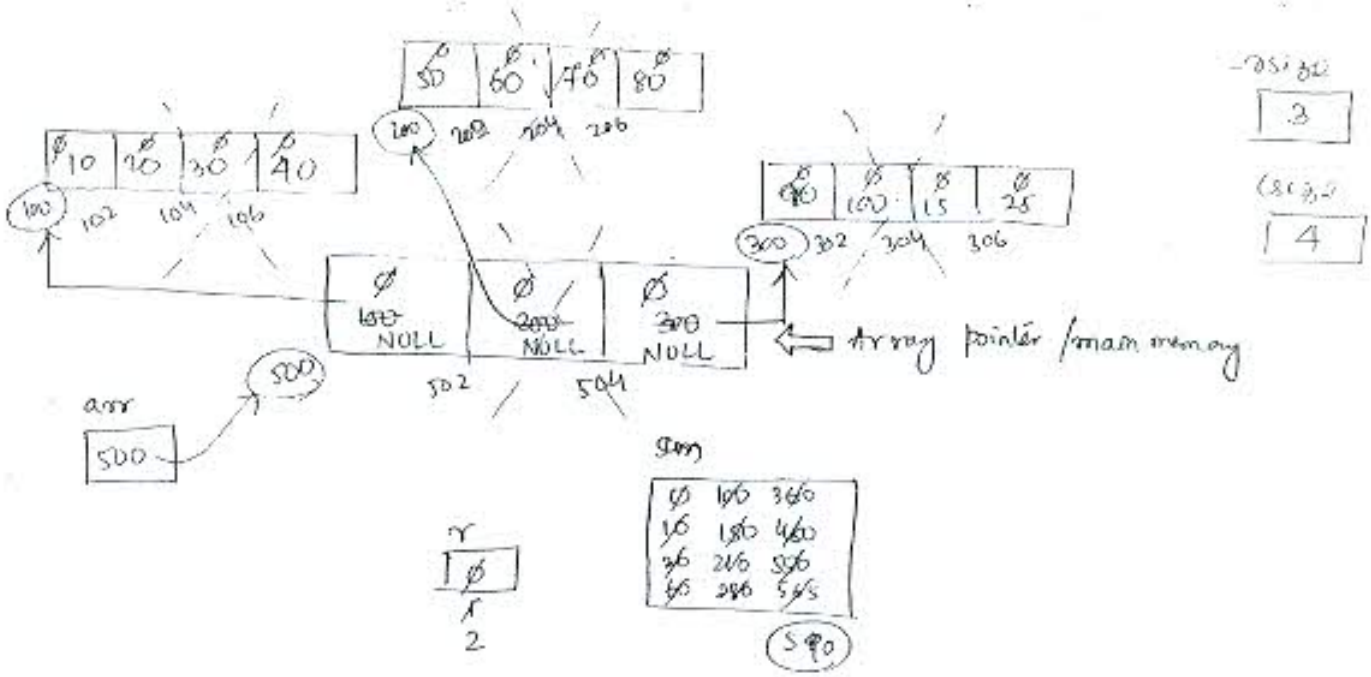
O/P

```
Enter size1 value : 5
Enter 5 values : 10 20 30 40 50
Enter size2 values : 3
Enter 3 values : 60 70 80
Arr List is : 10 20 30 40 50 60 70 80
```

0x4) /* 2-D DYNAMIC ARRAY */

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

void main()
{
    int **arr;
    int rsize, csize;
    int r, c, sum=0;
    clrscr();
    printf("Enter row size : ");
    scanf("%d", &rsize);
//  arr=(int *)calloc(rsize, sizeof(int)); 1D-Array
    arr=(int **)calloc(rsize, sizeof(int)); // Array pointer
    printf("\n Enter Column size : ");
    scanf("%d", &csize);
    for(r=0; r<rsize; r++)
        arr[r]=(int *)calloc(csize, sizeof(int)); // Array
    printf("\n Enter %d * %d values : ", rsize, csize);
    for(r=0; r<rsize; r++)
    {
      for(c=0; c<csize; c++)
          scanf("%d", &arr[r][c]);
    }
    for(r=0; r<rsize; r++)
    {
      for(c=0; c<csize; c++)
      {
          sum=sum+arr[r][c];
        // sum=sum+*(*(arr+r)+c)
      }
    }
    for(r=0; r<rsize; r++)
    {
        free(arr[r]);
        arr[r]=NULL;
    }
    free(arr);
    arr=NULL;
    printf("Sum : %d", sum);
    getch();
    return 0;
}
```



o/p

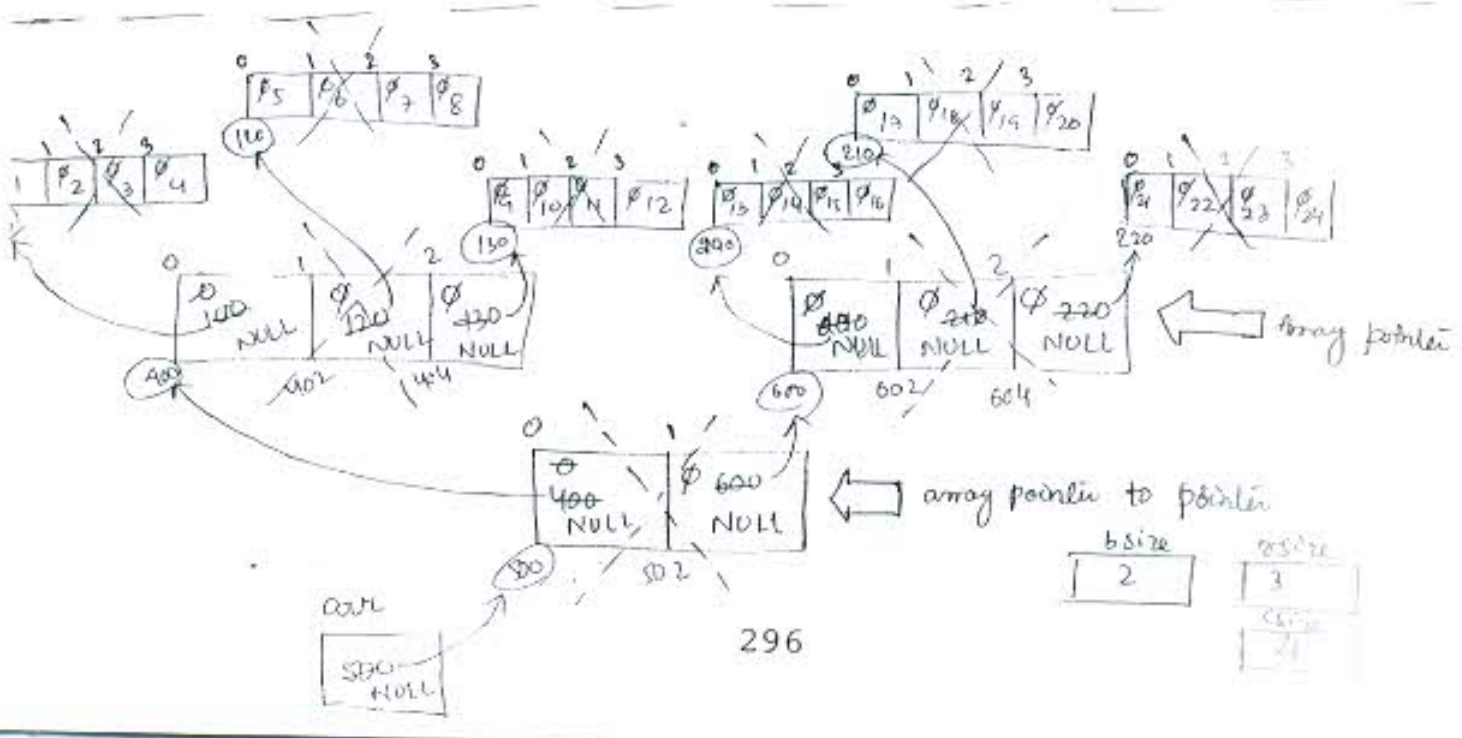
```
Enter row size : 3
Enter column size : 4
Enter 3 * 4 values :
10 20 30 40
50 60 70 80
90 100 15 25
Sum : 590
```

Qs) /* 3D ~~AP~~ DYNAMIC ARRAY */

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
int main()
{
   int ***arr;
   int bsize, rsize, csize;
   int b, r, c;
   clrscr();
   Printf("Enter block size : ");
   Scanf("%d", &bsize);
   arr = (int ***) calloc(bsize, sizeof(int)); // Array pointer to pointer
```

```
printf ("\n Enter Row size : ");
scanf (" %d", &rsize);
for ( b = 0; b< bsize ; b++ )
    arr[b] =(int **) calloc (rsize, sizeof (int));
printf ("\n Enter Column size : ");
scanf(" %d", &csize);
for(b= 0 ; b < bsize ; b++ )
for (r=0; r < rsize ; r++ )
        arr[b][r] = (int *) calloc (csize, sizeof (int));
printf (" Enter %d x %d x %d values ", bsize, rsize, csize);
for(b=0; b < bsize ; b++)
for (r = 0; r< rsize; r++ )
for(c=0; c< csize; c++ )
        scanf(" %d", &arr[b][r][c] );
for (b=0; b<bsize ; b++)
{
   printf ("\n Block : %d", b+1)
   for ( r=0; r < rsize ; r++ )
   {
     printf ("\n"),
     for ( c=0, c<csize ; c++ )
         printf (" %d", arr[b][r][c]);
         // printf("%d", *(*(*(arr+b)+r)+c));
   }

}
getch ();
```



```
// free the resources
for (b = 0 ; b < bsize ; b++)
for (r = 0 ; r < rsize ; r++)
{
    free (arr[b][r]);
    arr[b][r] = NULL ;
}
for( b = 0 ;  b < bsize ; b++)
{
    free (arr[b]);
    arr[b] = NULL ;
}
free (arr);
arr = NULL ;
return 0 ;
}
```

Output

```
Enter Block size : 2
Enter Row size : 3
Enter Column size : 4
Enter 2 * 3 * 4 values :
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
BLOCK : 1
1 2 3 4
5 6 7 8
9 10 11 12
BLOCK : 2
13 14 15 16
17 18 19 20
21 22 23 24
```

command line arguments :-

- It is a concept of passing the arguments to the main() function by using command prompt.
- when we are ~~working~~ designing the program for the DOS, then it is required to go for command line argument.
- DOS is a command interface O.S. i.e. if we need to execute any program we required to use command.
- By using command line arguments, we can create our own command, i.e. userdefined command.
- when we are designing the program on command line arguments, then main() function requires two arguments, i.e. "argc" and "argv".
- 'argc' is a variable of type an integer, it is a value type data which holds total no. of arguments count value.
- 'argv' is a variable of type char *, i.e. character array pointer, which maintains actual values, which are passed from command prompt.
- By default in 'C/C++' total no. of arguments count value is '1' i.e. 'Programname.exe'; means if the program name is '1.C' then command is "1.exe".

note) Command line related program always recommended to design by using commands only. Because if we are using TDE then we can compile and run the program but we can't pass the arguments to the main.

Steps to design command line Based Program :-

- To edit the program we required to use edit command.

Syntax) `edit filename.c`

ex:- E:\> edit myCmd.c

Code in myCmd.c :-

```
#include <stdio.h>
int main ( int argc , char * argv[] )
{
    int i;
    Printf ("\n Total no. of arguments : %d", argc) ;
    for (i=0 ; i<argc ; i++)
            Printf ("\n %d - Argument : %s ", i+1, argv[i]) ;
    return 0;
}
```

E:\> save myCmd.c (file is saved)

E:\> exit myCmd.c (exit from edit)

- To compile and link the program, we require to use 'TCC' command.

Syntax → `TCC filename.C`

Ex :- E:\> TCC myCmd.C

- When the compiling & linking is completed, then automatically, we will get an executable file i.e. myCmd.exe.

note: Executable filename itself is DOS command.

- To execute the program on command, we require to use command name or program name.exe.

- ex :- E:\> myCmd

```
Total no of arguments : 1
1. Argument : E:\myCmd.exe
```

ex :- E:\> myCmd 10 Hello 20 welcome

```
Total no of Arguments : 5
1. Argument : E:\myCmd.exe
2. Argument : 10
3. Argument : Hello
4. Argument : 20
5. Argument : welcome
```

note: 'argc' and 'argv' are local variables to the main function because according to the storage classes any function parameters are auto

- 'argc' and 'argv' are local variables to the main function, so we can't access command prompt related data outside of the main function.

- In implementation, when we need to access command prompt related data outside of the main() function then go for '_argc', '_argv' variables.

- '_argc' and '_argv' are global variables to DOS environment, which is declared in "dos.h".

- By using '_argc' and '_argv' variables we can access command prompt related data in any function.

Ans>
```
#include <stdio.h>
#include <dos.h>
void abc()
{
    int i;
    printf("\n Data in abc : ");
    printf("\n Total no of arguments : %d", -argc);
    for(i=0; i<-argc; i++)
        printf("\n %d. Argument : %s", i+1, -argv[i]);
}
int main(int argc, const char *argv[])
{
    int i;
    printf("\n Data in main : ");
    printf("\n Total no of arguments : %d", argc);
    for(i=0; i<argc; i++)
        pf("\n %d. Argument : %s", i+1, argv[i]);
    abc();
    return 0;
}
```

↳ Save as testcmd.C

↳ Compile & link using 'tcc' command.

E:\> testCmd Hello 10 welcome

```
Data in main :
Total no of Arguments : 4
1. Argument : E:\TestCmd.exe
2. Argument : Hello
3. Argument : 10
4. Argument : welcome

Data in abc :
Total no of arguments : 4
1. Argument : E:\TestCmd.exe
2. Argument : Hello
3. Argument : 10
4. Argument : welcome
```

(100) → When we are working with command line arguments then what type of data we are passing those all are treated as string type only. So when we are passing numeric type data then we require to convert string to numeric by using conversion related library functions.

→ Conversion related all library function are declared in <stdlib.h>.

① atoi() :-

- By using this predefined function, we can convert a string datatype to an integer type.
- atoi() function requires one argument of type const char * and returns an integer value.

Syntax → `int atoi(const char *str);`

② atol() :-

- By using this predefined function, we can convert a string to long integer value.
- atol() function returns long int type and the parameter type is const char *.

Syntax → `long atol(const char *str);`

③ atof() :-

- By using this predefined function we can convert a string to float type.
- atof() function takes one argument of type const char * and returns double type.

Syntax → `double atof(const char *str);`

④ _atold() :-

- By using this predefined function, we can convert a string to long double type.

Syntax → `long double _atold(const char *str);`

- By using ecvt(), fcvt() & gcvt() function, we can convert a float data into string type.

Eg)
```
#include <stdio.h>
#include <stdlib.h>
int myatoi(const char *str)
{
    int n=0, i=0, sign=0;
    if(str[0]=='+' || str[0]=='-')
    {
        i=1;
        if(str[0]=='-')
            sign=1;
    }
```

```
for( ; str[i] != '\0' ; i++)
{
    if (str[i] > '0' && str[i] <= '9')
        r = r*10 + str[i] - '0';
    else
    { if (sign == 1)
        return (r * -1);
      else
        return (r);
    }
}
if (sign == 1)
    return (r * -1);
else
    return (r);
}
int main (int argc, const char * argv[])
{
    int i, n ;
    if (argc != 2)
    {
        Printf("\n Invalid arguments ");
        return 1 ;
    }
// r = atoi (argv[1]);
    r = myatoi (argv[1]);
    for(i = 1 ; i<=10; i++)
        Printf ("\n %d * %d = %3d ", n, i, n * i);
    return 0;
}
```

22 Nov

STRUCTURES

- Structure is a collection of different types of data elements in a single entity.
- In 'C' programming language, data types are classified into 3 types :-
  1. primitive data types
  2. derived data types
  3. user defined data type.
- All primitive and derived data types are designed to work for basic data types. i.e. char, int, float types.
- In real time world, every information will be there in object format.
- Every object contains their own properties and behaviour.
- No any predefined and derived datatypes support real time object data model.
- In implementation, whenever primitive & derived datatypes are not supporting user requirements, then go for user-defined datatypes.
- By using structures, we can create our own user defined data types.
- A structure is a combination of primitive & derived data type variables.
- 'C' prog. lang. structures contain data members only, in C++ data members & member functions both are present.
- The size of the structure is the sum of all member variable size.
- least size of the structure is 1 byte.
- In 'C' prog lang., it is not possible to create empty structure, but in C++ it is possible.

- Syntax

```
struct tg_name
{
    Datatype1  mem1
    Datatype2  mem2
    Datatype3  mem3
    ....
};
```

- When we are working with the structures, at end of the structure body semicolon (;) must be required.
- Ex:-

```
struct emp
{
    int id;
    char name[36];
    int sal;
};
```

Size of (struct emp) = 40 B

Ex 2) struct student

```
{
    int r no;
    char sname [20];
    char fname [20];
    int marks;
    int tmarks;
    float avg;
};
```

sizeof (struct student) ⇒ 60 bytes

Syntax to create structure variables :-

```
struct tag_name variable_name ;
```

```
struct emp
{
    int id;
    char name [36];
    int sal;
};
void main()
{
    emp e1 ; // Error
    struct e1; // Error
    struct emp e1 ;
    struct emp e2, e3, e4 ;
};
```

struct size → 40 B

Structure variables

sizeof(e1) → 40 bytes ← var. struct size

- When we are creating the structure variable, it can be created at the end of the structure body also, that is before semicolon (;), we require to place

eg:

```
struct emp {
    int id;
    char name [36];
    int sal;
} e1, e2 ;    // e1, e2 are structure variables.

void main()
{
    struct emp e3, e4 ;
}
```

- when we are constructing structure variable at the end of the struct body, then it becomes global variables - e1, e2 ;
- when we are constructing the structure variables within the body then those are become auto variables , ex :- e3, e4 .

- Syntax to create structure pointers :-

```
struct tagname * pointer name;
```

- ex) struct emp
{
int id;
char name[20];
int sal;
};
void main()
{
struct emp e1;
struct emp *ptr;
ptr = &e1;
}

note) sizeof(e1) → 24 B
sizeof(ptr) → 2B

ex2) struct emp
{
int id;
char name[36];
unsigned int sal;
} e1, *ptr1; // global ptr
void main()
{
struct e2 , *ptr 2 ; // local ptr
ptr 1 = &e1 ;
ptr 2 = &e2 ;
}

* Syntax to create structure type array :-

```
struct tagname arr[size]
```

```
struct emp
{
int id;
char name[36];
int sal;
}
```

```
void main ()
{
struct emp arr[10];
}
```

size of 30 → 10
size of arr → 400B

Syntax to create Structure array dynamically :-

```
struct emp
{
   int id;
   char name[36];
   int sal;
};
void main()
{
   struct emp *arr;
   arr = (struct emp*) calloc(10, sizeof(struct emp));
   - - -
   - - -
   free(arr);
   arr = NULL;
}
```

Syntax to initialize Structure variables :-

```
struct tagname variable = { mem1 = value, mem2 = value2, ... };

struct emp
{
   int id;
   char name[36];
   int sal;
} e1 = {1, "Saumya", 22500},
  e2 = {2, "Bapun", 23600};
void main()
{
   struct emp e3 = {"BPB"}, e4, e5;
}
```

On initialization of the structure variable, if specific no. of members are not initialized then remaining elements will be initialized with '0' or NULL.

3) If numeric field is not initialized, then value becomes '0', if string field is not initialized then value becomes NULL.

Syntax to access structure members :-

- By using following operators, we can access structure members :-
  - struct. to member
  - → pointer to member

- If the variable is normal type, then go for struct. to member operator, if the variable is pointer type, then go for pointer to member operator.

e.g. ① struct tagname variable;
variable.member = value; // assign
variable.member; // access

② struct tagname *ptr;
ptr → member = value; // assign
ptr → member; // access.

e.g.
```
struct emp
{
    int id;
    char name[36];
    int sal;
} e1 = {1, "Saranya", 20000},
  e2 = {2};
void main()
{
    struct emp e3, e4, e5;
    e3.id = 3;
    e3.name = "SPB";
// e3.name = SPB; //error
//strcpy(e3.name, "SPB"); ✓
    e3.sal = 45000;
    strcpy(e2.name, "Bapu");
    e2.sal = 22000;
    e4.id = e3; //err
    e4 = e3.id; //err
    e4.id = e3.id + 1;
    strcpy(e4.name, e1.name);
    strcat(e4.name, e2.name);
```

```
e4.sal = e1 + e2; //error
e4.sal = e1.sal + e1.sal; ✓
e5 = e3; // both should be same struct var
//e2.id > e1.id //yes
e4.sal = = e3.sal; //yes
e1.name = = e2.name; //error
strcmp(e3.name, e2.name); //yes
}
```

- On structure variable, except assignment operator, no any other operations are allowed.
- One structure variable data can be assigned to another variable, if both variables are same structure type.
- Any kind of operations can be allowed on property of a structure variable.

ex:
```
struct emp
{
    int id;
    char name[36];
    int sal;
};
void main()
{
    struct emp e1;
    struct emp * ptr;
    ptr = &e1;
    e1.id = 1; // yes
    ptr -> id = 1; // yes
    // e1 -> id = 1; // err
    // ptr.id = 1; // err
    strcpy(e1.name, "Krish");
    strcpy(ptr -> name, "Krish");
    e1.sal = 15000;
    ptr -> sal = 15000;
}
```

→
```
struct emp
{
    int id = 1;
    char name[36] = "Raj";
    int sal = 12500;
}; // error
```

- For declaring a structure, it doesn't occupy any physical memory, so we can't initialize the data within the body.
- memory will be constructed for structure, when we are creating variable.

→ struct emp

```
{
};
```

ERROR

Self Referential structure :-

- Placing a structure type pointer as a member to same structure, it is called self referential structure.
- Complete data structures will be implemented by using self referential structure only.

→ struct emp

```
{
    int id;          2
    char name[36];   36
    int sal;         2
    struct emp *ptr; 2
};
```

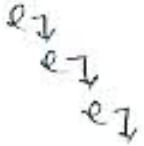
sizeof(struct emp) → 42 Bytes.

✓ It is not possible to place structure type variable as a member to same structure, because logically the size of the structure becomes infinite.

i.e.

```
struct emp
{
    int id;
    char name[36];
    int sal;
    struct emp e;
};
// ERROR
```

infinite



3/11/12

typedef

{ #define → danger on 1st zone
* typedef → danger on 2nd zone

- It is a keyword, by using this keyword, we can create an user defined name for existing classes.
- Generaly this keyword is required to use, when we need to create an alias name for an existing data type.

- Syntax: typedef datatype userdefinedname;

ex:>
```
typedef int smallint;
void main()
{
  int a=10;
  smallint b=20;
  typedef smallint myint;
  myint c;
  c=a+b;

  Printf("%d + %d = %d", a, b, c);
  getch();
}
```

o/p: 10 + 20 = 30

- In the above program, smallint is a user defined name for existing datatype 'int'.
- myint is a user defined name for existing user defined name
- All the properties of integers are applicable to smallint & myint also.

ex:>
```
typedef char byte;
#define mychar char;
void main()
{
  byte b='B';
  mychar ch1='D';
  char ch2;
  ch2 = ch1 - b + 65;
  printf("%c %c %c", b, ch1, ch2);
}
```

o/p: B D C

- By using typedef only, we can create an alias name & it is under the control of compiler.
- By using '#define', we can't create alias names, because where an identifier is occurred, it is replaced with replacement text. So we can't create user defined names.
- When we are working with the structures, mentioning the tag name is always optional.
- If the tag name is not available, then compiler will creates a nameless structure.

e.g.
```
struct
{
    int id;
    char name[36];
    int sal;
};   // valid
```

- When we are working with nameless structures, then it is possible to create global variables only, because for creation of the local variable, we require tagname also.

ex1>
```
struct
{
    int id;
    char name[36];
    int sal;
} c1, c2;
void main()
{
    struct tagname ?  // local declaration
}
```

ex2>
```
struct emp
{
    int id;
    char name[36];
    int sal;
};
typedef struct emp Emp;
void main()
{
    emp e1;  //error
    struct emp e1;  //yes
    Emp e2;  //yes
}
```

→ In this syntax, compiler is creating an alias name for struct emp as Emp.

ex3) typedef struct {
    int id;
    char name[36];
    int sal;
} Emp;
void main() {
    Emp e1; // valid
}

- In the above syntax, first compiler creates a nameless structure, for that nameless structure, it creates an alias name called Emp.

ex4) typedef struct emp
{
    int id;
    char name[36];
    int sal;
} Emp;
void main() {
    Emp e1;
    struct emp e2;
} // valid

⇒ In this case, compiler will create struct emp as a userdefined datatype, for that userdefined datatype it creates an alias name called "Emp".

ex5) struct emp{
    int id;
    char name[36];
    int sal;
} e1, e2, e3; // global var.

typedef struct emp {
    int id,
    char name[36];
    int sal;
} e1, e2, e3; ← alias names

- when the structure is started with 'typedef' keyword, then it's not possible to create global variables at end of the structure body.

ex6) typedef struct emp {
    int id;
    char name[36];
    int sal;
} Emp e1, e2, e3; // error.

ex7)
```
typedef struct emp {
        int id;
        char name[36];
        int sal;
} Emp, e1, e2, e3; // valid
```
(Emp: alias name)

ex8)
```
struct emp{
    int id;
    char name[36];
    int sal;
};
void abc() {
    struct emp e1, e2;
}
void main()
{
    struct emp e1, e2;
} //valid.
```

ex9)
```
void main()
{
    struct emp
    {
        int id;
        char name[36];
        int sal;
    };
    struct emp e1;
}
```

```
void abc()
{
    struct emp e2;
}
```
(ERROR)

26/11/20

File-operation

- A file is a name of the physical memory location within secondary storage area, that is hard disk.
- In implementation, when we need to read or write the data from the secondary storage area, then go for file-operations.
- By using file operations application related data can be stored permanently in the secondary storage area.
- Whenever we require to load the information from secondary storage area, then also we can use file-operations.
- In C-programming language, I/O operations are classified into 2 types, that is
  1. Standard I/O operation
  2. Secondary I/O operation.
- When we are interacting with standard IO devices, then it is called Standard I/O operations. When we are interacting with secondary I/O devices, then it is called secondary I/O operation.
- Standard I/O related / secondary I/O related, all predefined functions are declared in "stdio.h".
- File operated predefined functions are :-

<stdio.h> :-

fopen	fclose	fprintf	fputs
fscanf	fgetc	fgets	fgetchar
fread	fputc	flushall	ftell
feof	fseek	remove	rename.
rewind	fflush		

constant	datatype	global variables
EOF	FILE	stdin
NULL	↓	stdout
SEEK_CUR		stderr
SEEK_END		stdprn
SEEK_SET		

Ex)
```
#include
#
int main()
{
    FILE *fp;
    fp=fopen("h:\\1.txt","w");
    if(fp==NULL)
    {
      printf("\n FILE NOT CREATED");
      getch();
      return 1;
    }
```
insertion → `fprintf(fp,"welcome");`
→ `fprintf(fp,"\n%s%d",hello,20);`
save & exit → `fclose(fp);`
→ `return(fp);`
```
}
```

FILE is a predefined structure alias name.

fopen < null / add

- "stdio.h" provides standard i/o related all predefined function prototypes.
- "conio.h" provides console related all predefined function prototype.
- 'FILE' is a predefined structure, which is defined in <stdio.h>.
- By using FILE structure, we can handle file properties.
- The size of the file structure is 16 Bytes.
- fp is a variable of type FILE *, by using this pointer we can handle file address.

① fopen() :-

- It is a predefined function, which is declared in "stdio.h", by using this function, we can open a file in specific path with specific mode.
- fopen() function requires 2 arguments of type constant char *, i.e. an address of a string.

```
FILE *fopen(const char *path, const char *mode);
```

- path will indicate location of the file. mode indicates for what purpose we open the file.

- On success fopen(); returns file*, on failure that is when the file is not open, it returns NULL.

Note) Generally, fopen() fails to open the file in following cases :-

1) path is incorrect.
2) mode is incorrect.
3) permissions don't exist.
4) memory is not available.

2) fprintf() :-

- It is a predefined function, which is declared in stdio.h.
- by using this predefined function, we can print the data onto the file.
- fprintf() can take only no of arguments, but first argument must be file* and remaining arguments are printf function format.

When we are working with fprintf(), it returns an integer value that is total no of characters, which is printed into file.

Syntax) int fprintf (FILE * stream, const char * format, ...);

3) fclose() :-

- It is a predefined function, which is declared in <stdio.h>, by using this predefined function, we can close the file, after saving the data.

fclose() requires one argument of type FILE * & returns an integer value

- when it returns 0 → file is closed properly.
  when it returns -1 → file is failed to close i.e file is not closed properly.

```
int fclose (FILE * stream);
```

Ex) ASCII to a text file

```
#
#
#include<process.h>
int main()
{
    FILE * fp;
    char path[30];
    int i;
    clrscr();
    printf("\n Enter path:");
```

```
gets(path);
fp = fopen(path, "w");
if (fp == NULL)
{
    printf("\n unable to create file");
    getch();
    exit(1); //return 1
}
for(i = -128; i <= 127; i++)
    fprintf(fp, "%d = %c\t", i, i);
fclose(fp);
return 0;
}
```

(note) when we are providing the path dynamically, if complete path is not provided, then compiler is going to create the file in current location only, that is in which location application is executing in same location, it will be created.

file modes :-

- Always file modes indicate that for what purpose file is opened. File modes are classified into 3 types that is :-
  - write - w
  - read - r
  - append - a
- Depending upon the operations, file modes are classified into 6 types

① w (wrote) :-

- creates a file for writing the data.
- If the file is already existing with that name, then it will override that is old file will be deleted and new file will be created with same name.
- In 'w' mode, if file is existing or not always new file will be created.

② r (read) :-

- opens an existing file for reading.
- In 'r' mode, if file is already existing, then it'll open for reading. if file is not exist then fopen returns null.
- In 'r' mode, file is exist or not, new file will not be constructed.

③ append (a) :-

- opens an existing file for writing, if file is not existed, then new file will be created for writing.
- In append mode, if file is not existing then only new file will be constructed.

④ w+ (write and read) :-

- Creates a file for updating that is writing & reading.
- In w+ mode, if the file already exists, then also it will create new file, i.e. old file will be deleted.
- In w+ mode, if file is existing or not, always new file will be created.

⑤ r+ (read & write) :-

- Opens a existing file for updating, i.e. reading & writing.
- In r+ mode, if file is existing or not, new file will be not be constructed.

⑥ a+ (w+ and r+) :-

- open an existing file for updating

or

creates a new file for updating.

- In a+ mode, if file doesn't exist, then only new file will be constructed.

In C-prog. language, files are classified into 2 types, i.e.

1) Text file
2) non Text file

- In text files, data will be represented with the help of ASCII values.

.txt, .doc, .c, .cpp, .xls .. etc.

- In binary files, dat will be represented in Bytes format.

.jpeg, .png, .mp3, .mp4, .vob.

① To specify that a given file is being opened or created in "Text mode" then append 't' to the string mode.

ex:- rt, wt, at, r+t, w+t, a+t.

② To specify binary mode, append 'b' to end of the string mode.

rb, wb, ab, r+b, w+b, a+b.

③ "fopen" and "_fsopen" also allow that "t" or "b" to be inserted between the letter and 't' character in the string mode.

r t + is equivalent to r + t .

④ If 't' or 'b' is not given in the string, mode is governed by '_f' mode, if '_f' mode is set to O_BINARY, files are opened in binary mode.

⑤ If _f mode is set to O_TEXT, they are opened in text mode, these constants are defined in "FCNTL.H".

General, if we are not specifying Text/Bi

{ O_TEXT
  O_BINARY

1/11/12

file read op'n from harddisk :-

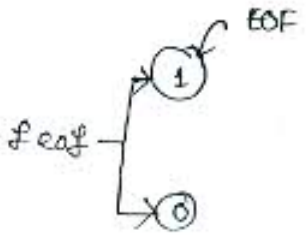
ex1)

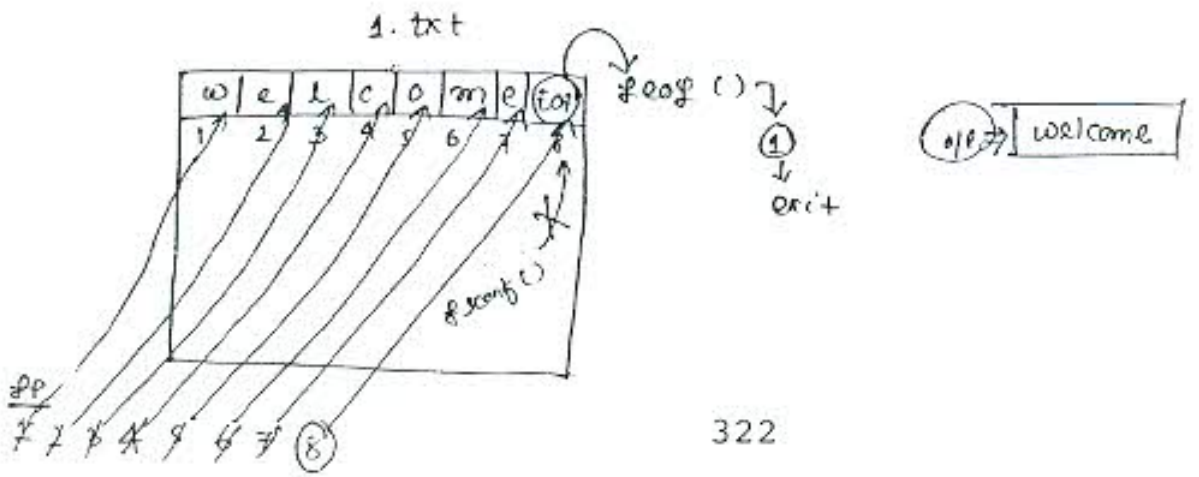
```
#include <stdio.h>
#include <conio.h>

int main() {
    FILE *fp;
    char ch;
    fp = fopen("h:\\1.txt","r");
    if (fp == NULL)
    {
        printf("\n FILE NOT FOUND");
        getch();
        return 1;
    }
    while (1)
    {
      fscanf(fp, "%c", &ch);
      if (feof(fp)) // if(ch==EOF)
            break;
      printf("%c", ch);
    }
    fclose(fp);
    return 0;
}
```

fscanf() :-

reads the entire file data excluding EOF character.





fscanf() → It's a predefined function, declared in <stdio.h>.

- By using this function, we can read the data from file.
- fscanf() function takes any no of arguments, but first argument must be "FILE *" and remaining arguments are 'scanf()' function format.
- When we are working with 'fscanf()' function, then it can read the entire content of the file except 'EOF' character.

- Syntax:- `int fscanf ( FILE * stream, const char * format, .... );`

feof() →

- It's a predefined function, declared in "stdio.h", by using this function, one can find 'EOF' character position.
- feof() function requires one argument of type "FILE *" and returns an integer value.
- When the file pointer is pointing to 'EOF' character, then feof() function returns non-zero,

  Other than EOF character, it returns zero.

- Syntax :- `int feof ( FILE * stream);`

Ex2) Enter a file :- h:\cpp.txt

Suspend for ½ sec after a new line

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
int main()
{
    FILE *fp;
    char path[30]; char ch;
    clrscr();
    fprintf(stdout, "Enter file path :- ");
    // Printf (" Enter file path :- ");
    gets(path);
    fp = fopen ( path ,"r");
```

```
if (fp == NULL)
{
    fprintf( stderr , " FILE NOT FOUND");
    // printf (" FILE NOT FOUND"),
    getch();
}
while (1) {
    ch = fgetc(fp); // fscanf(fp, "%c", &ch);
    if(ch == EOF)     // if (feof (fp))
        break;
    if(ch != '\n')
        delay(500);
    printf(" %c", ch);
    // fprintf (stdout, "%c", ch);
}
fclose (fp);
getch();
return 0;
}
```

fgetc()

- It's a predefined function, declared in stdio.h, by using this function we can read a character from file.
- fgetc() function takes one argument of type "FILE *" and returns an integer value. The return value of fgetc() function is an integer, i.e ASCII value of read character
- Syntax:- `int fgetc (FILE* stream);`
- When we are working with fgetc() function, it can read entire content of file including 'EOF' character also.

delay()

- It's a predefined function, declared in "dos.h"; by using this function, we can suspend a program in millisecond format
- delay() function requires one argument of type unsigned integer.

Syntax:- `void delay (unsigned milliseconds)`

Appending to a file :-

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

int main(){
    FILE *fp;
    fp = fopen("h:\\1.txt","a");
    if(fp==NULL){
        printf("\n UNABLE to CREATE FILE");
        getch();
        return 1;
    }
    fprintf(fp, "\n Naresh IT");
    fprintf(fp, "\n %d NIT %d", 100, 200);
    fclose(fp);
    getch();
    return 0;
}
```

```
#
#
int main()
{
    FILE *fp;
    char str[50];
    char path[20];
    clrscr();
    fprintf(stdout, "Enter file path :-");
    fscanf(stdin, "%s", path);
    //scanf("%s", path);
    fp = fopen(path,"a");
    if(fp==NULL){
        printf("\n file not created ...");
        // fprintf(stderr, "file not created ...");
        getch();
        return 1;
    }
```

```
    printf("\n Enter a String:");
    fflush(stdin); // fflush(stdin);
    gets(str);
    fputs(str, fp);
    //fprintf(fp, "%s", str);
    fclose(fp);
    return 0;
}
```

fputs() :-

- It's a predefined function, used to print the string data in file
- fputs() function requires two arguments of type 'const char *', and 'FILE *'
- Syntax: int fputs(const char *str, FILE * Stream);

fgets() :-

- By using this function, we can read the string data from file.
- fgets() function requires three arguments of type char *, int, & FILE *
- Syntax: char * fgets(char *s, int n, FILE * stream);
- 's' holds an address of a string, 'n' indicates no of characters, & 'stream' is an address of the file.

11/12

```
/* Command creation */
#
int main(int argc, char * argv[])
{
    FILE * sfp, * dsp;
    char ch;
    if (argc != 3)
    {
        Printf("\n Command Syntax invalid");
        return 1;
    }
    sfp = fopen(argv[1], "rt");
    if (sfp == NULL){
        printf("\n % not found.", argv[1]);
        return 1;
    }
```

```
dfp = fopen( argv[2], "wt");
if ( dfp != NULL)
{
    do {
        printf("\n Over write  %s ? (Y/N) :", argv[2]);
        fflush all();
        ch = getchar();
    } while (ch != 'Y' && ch != 'y' && ch != 'N' && ch != 'n');
    if ( ch == 'N' || ch == 'n'){
        printf("\nO file(s) copied");
        fclose(sfp);
        fclose(dfp);
        return 0;
    }
    else {
        fclose (dfp);
    }
```

~~dfp = fopen(argv[2],"wt");~~

```
if(dfp == NULL) {
    printf("\nUnable to Copy");
    fclose( sfp);
    return 1;
}
while (1) {
    ch = fgetc(sfp);
    if (ch == EOF)
        break;
    fprintf( dfp, "%c", ch);
}
fclose (sfp);
fclose (dfp);
printf("\n 1 file(s) coppied. ");
return 0;
} // save as mycopy.c
// compile & link using TCC.
```

stdin

- 'stdin' is a global pointer variable, which is declared in stdio.h, by using pointer variable, we can handle standard input buffer.

stdout

- It is a global pointer variable declared in stdio.h, by using this pointer variable, we can handle standard output buffer.

stderr

- By using this pointer variable, we can handle standard io related errors and that error message will redirect back to stdout.

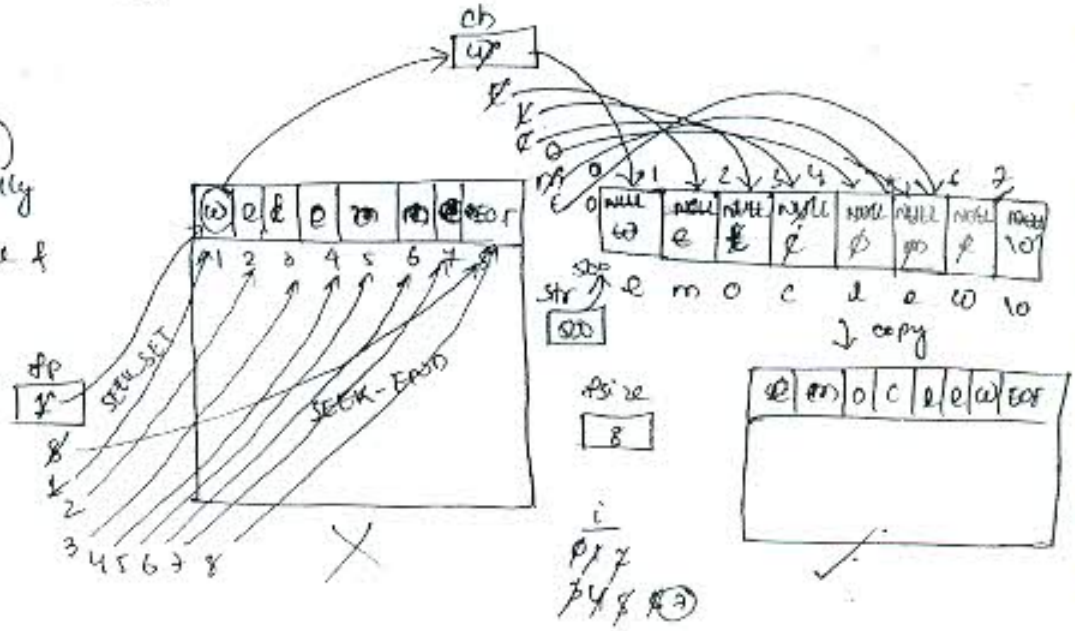
stdprn

- By using this pointer variable, we can handle standard printer.

ex6) /* file content reversing */

steps

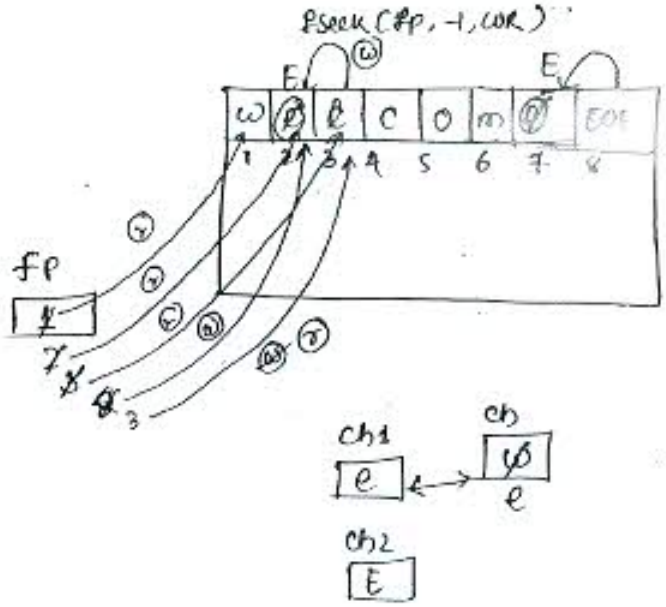
1) open file for reading
2) find size of file
3) create string dynamically
4) read data from the file & copy to string.
5) strrev(string)
6) close file / opening in write




```
#include <stdio.h>
#include <alloc.h>
#include <string.h>
int main()
{
    FILE *fp;
    char *str;
    char ch;
    long int fsize, i=0;
    char path[30];
    clrscr();
```

```
printf ("\n Enter file path:");
gets (path);
fp = fopen (path, "r+");
if (fp == NULL){
        printf("\n FILE NOT FOUND");
        getch();
        return 1;
}
//moving fp to END_of_File)
fseek (fp, 0, SEEK_END);
//find the position of fp.
fsize = ftell (fp);
fseek (fp, 0, SEEK_SET); // Passing fp to beginning of file.
str = (char *) calloc (fsize, sizeof (char));
while (1)
{
    ch = fgetc(fp);
    if (ch == EOF)
        break;
    str [i++] = ch;
}
str [i] = '\0';
fclose (fp);
remove (path);
strrev (str);
fp = fopen (path," wt " );
fputs (str, fp);
// fprintf (fp," %s", str);
fclose (fp);
free (str);
str = NULL;
return 0;
}
```

(17) /* Replacing a specific character by another character in a file */



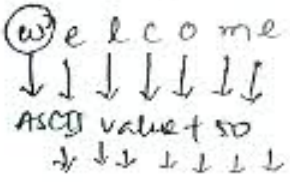

```
#include <stdio.h>
#include <conio.h>
int main()
{
   FILE * fp;
   char ch1, ch2, ch;
   char fname[30];
   clrscr();
   printf("\n Enter file name : ");
   gets(sname);
   fp = fopen(fname, "r+");
   if(fp==NULL){
         printf("\n FILE NOT FOUND");
         getch();
         return 1;
   }
   printf("\n Enter char 1 (s) : ");
   flushall();
   ch1 = getchar();
   printf("\n Enter char 2 (r) : ");
   fflush(stdin);
   ch2 = getchar();
   while(1){
      ch = fgetc(fp);
      if(ch==EOF)
            break;
      if(ch==ch1){
            fseek(fp, -1, SEEK_CUR); // pos 'fp' to 1 byte back pos.
            // read → write by -1 pos
            fprint(fp, "%c", ch2);
            fseek(fp, 0, SEEK_CUR); // w → r by '0'
      }
   }
   fclose(fp);
   return 0;
}
```

25/1/12

ex> /* file corrupting program */

#
#

```
int main(){
    FILE *fp;
    char ch;
    int code, option;
    char fname[36];
    clrscr();
    printf("Enter file name : ");
    gets(fname);
    fp = fopen(fname, "r+t");
    if(fp==NULL){
            printf("FILE NOT FOUND");
            getch();
            return 1;
    }
    do{
      printf("\n For encoding press 1.");
      printf("\n For decoding press 2.");
      printf("\n Enter option : ");
      scanf("%d", &option);
    }while(option!=1 && option!=2);
    if(option==1)
          code=50;
    else
       code=-50;
    while(1){
        ch = fgetc(fp);
        if(feof(fp))
             break;
        if(ch!='\n' && ch!='\r'){
                fseek(fp,-1, SEEK_CUR); // r → w by -1
                fprintf(fp, "%c", ch+code);
                fseek(fp, 0, SEEK_CUR); // w → r
        }
    }
    fclose(fp); return 0;
}
```



fseek()

- By using this predefined function, we can rearrange the file pointer position.
- fseek() function requires 3 arguments of type 'FILE *', 'long int', & 'int' type arguments.

```
int fseek(FILE * stream, long offset, int whence);
```

- stream will provide file pointer location, offset value is no. of bytes, whence is position of the file pointer.
- File pointer positions can be recognized by the following constant values:-

  Such as:-

  1. SEEK_SET : It provides file pointer position to beginning of the file. SEEK_SET equivalent int value is zero (0).
  2. SEEK_CUR : gives current file pointer position, equivalent int value is '1'.
  3. SEEK_END : gives the end posn or pass the file pointer to end of the file. - equivalent int value is '2'.

ftell()

- By using this predefined function, we can find size of the file.
- ftell() functions requires one argument of type 'FILE *' and returns long int value.
- ftell() function gives file pointer position, i.e. no. of bytes travelled by file pointer; i.e. equivalent to size of the file.
- Syntax:-

```
long ftell(FILE * stream);
```

rewind()

- By using this predefined function, we can pass the file pointer to beginning of the file.
- rewind() function requires one argument of type FILE *.
- The behaviour of the rewind() function is similar to:

```
fseek(FILE * stream, 0, SEEK_SET);
```

- Syntax:-

```
void rewind(FILE * stream);
```

rename()

↳ By using this function, we can change the name of the file.

↳ rename() function requires two arguments of type 'const char *' & 'const char *'.

↳ syntax:- int rename (const char * oldname, const char * newname);

↳ on success, rename() function returns '0', on failure it returns '-1'.

↳ ex

remove()

↳ By using this predefined function, we can remove the file.

↳ remove() function requires one argument of type FILE *, & returns an int value.

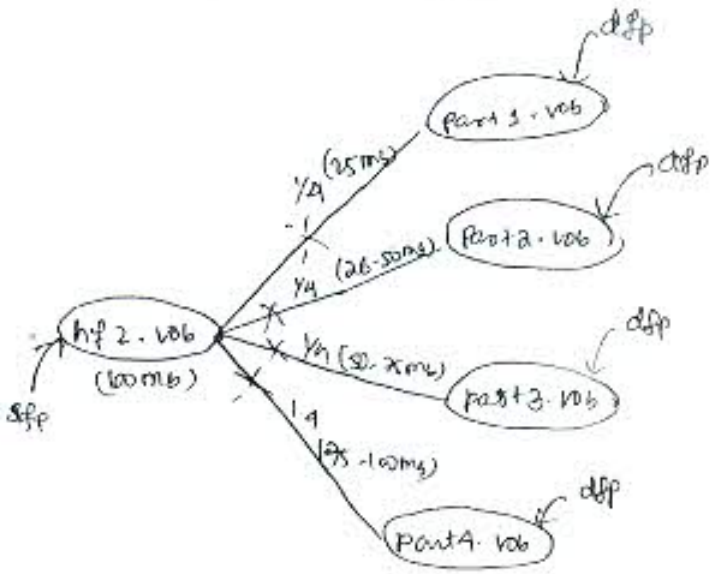
↳ On success, it returns '0' & on failure, it returns '-1'.

↳ syntax:- int remove (const char * filename);

ex) /* file cutting / file splitting program */

```
# include
# include
# define f1 "h:\\ part1. vob"
# define f2 "h:\\ part2. vob"
# define f3 "h:\\ part3. vob"
# define f4 "h:\\ part4. vob"
int main(){
    FILE *sfp, *dfp;
    long int fsize, nb=0;
    int i=0; char ch;
    char sfname[30];
    char dfname[4][30] = {f1, f2, f3, f4};
    clrscr();
    printf("\n Enter source file:");
    gets(sfname);
    sfp = fopen(sfname, "rb");
    if (sfp == NULL){
        printf("\n %s not found", sfname);
        getch();
        return 1;
    }
```

```
dfp = fopen ( dfname [0], "wb");
if (dfp == NULL){
        printf ("\n Unable to Create % s.", dfname [i]);
        fclose (sfp);
        getch ();
        return 1;
}
while (1)
{
    fseek ( sfp , 0 , SEEK_END);
    fsize = ftell (sfp);
    fseek( sfp, 0, SEEK_SET ); // rewind (sfp);
    ch = fgetc (sfp);
    if (feof (sfp))
        break ;
    fprintf ( dfp , "%c", ch) ;
    ++nb ;
    if (nb == fsize/4 && i != 3)
    {
        fclose (dfp);
        nb=0 ;
        dfp = fopen( dfname[++i] , "wb");
        if (dfp==NULL)
        {
            printf("\n %s Unable to Create.", dfname [i]);
            fclose (sfp);
            getch () ;
            return 1;
        } // inner if
    } // outer if
} // while
fclose (sfp);
fclose (dfp);
return 0;
}
```



/* file merging program */

ex2)

```
#
#
#define f1 "h:\\ part1. mp3"
# define f2 "h:\\ part2. mp3"
# defde f3 "h:\\ part3. mp3"
# defde f4 "h:\\ part4. mp3"
int main(){
   FILE *sfp, *dfp;
   char ch ;
   int i ;
   chardfname[30];
   char sfname[4][30]: { f1, f2, f3, f4};
   clrscr();
   printf("\n Enter Test File : ");
   gets( dfname);
   dfp = fopen (dfname , "wb");
   if ( dfp == NULL){
            printf("\n Unable to Create %s", dfname);
            getch();
           return 1;
    }
    for(i=0; i<4; i++) {
        sfp= fopen (sfname[i], "rb");
        if (sfp==NULL) {
              printf("\n %s not found", sfname[i]);
              fclose (dfp);
              getch();
        }
        while (1) {
            ch= fgetc (sfp);
            if(feof (sfp))
                 break;
            fprintf(dfp, "%c", ch);
        }
        fclose (sfp);
   }
   fclose(dfp);
   return 0;
}
```

DLL

ex1)

```
#
#
# include <process.h>
# define oldfile "E:\\ C900dm OLD.dat"
# define curfile " E:\\ C900dm TMP.dat"

typedef struct
{
    int id;
    char name[36];
    unsigned int sal;
    float pf;
} Emp;

Emp e;
FILE *cur, *old;

int olddata()
{
    int ne;
    old = fopen(oldfile, "rb");
    if(old == NULL)
        return 0; // no records
    fseek(old, 0, SEEK_END);
    ne = ftell(old)/sizeof(Emp);
    fclose(old);
    return ne;
}
int curdata()
{
    int ne;
    cur = fopen(curfile, "rb");
    if(cur == NULL)
        return 0;
    fseek(cur, 0, SEEK_END);
    ne = ftell(cur)/sizeof(Emp);
    fclose(cur);
    return ne;
}
```

```
void createemp()
{
    char ch;
    int ne;
    ne = curdata() + oldata();
    cur = fopen(curfile, "ab");
    if (cur == NULL)
    {
        printf(" UNABLE TO CREATE DATA BASE ");
        getch();
        exit(1);
    }
    do{
        e.id = ++ne;
        printf("\n Emp Id : %d", e.id);
        Printf ("\n Enter Emp Name : ");
        fflush(stdin);
        gets(e.name);
        printf("\n Enter EMP salary : ");
        scanf("%d", &e.sal);
        e.pf = (float) e.sal * 10/100;
        fwrite(&e, sizeof(EMP), 1, cur); // application data to file
        printf("\nDo You want to Continue y? ");
        flushall();
        ch = getchar();
    } while (ch == 'Y' || ch == 'y');
    fclose(cur);
    return 0; // return back to calling location, i.e. main
}
```

```
void shaoold()
{
   int ne, i;
   ne = oldata();
   if(ne==0)
   {
      printf("\n ***** NO RECORDS IN OLD LIST ***** ");
      getch();
      return;
   }
   old= fopen(oldfile, "rb");
   if (old == NULL)
   {
      printf("\n UNABLE TO READ OLD LIST ");
      getch();
      return 0;
   }
   printf("\n -------- OLD RECORD LIST -------- ");
   for(i=1; i<=ne; i++)
   {
      fread( &e, sizeof(EMP), 1, old ); // file data to application
      printf("\n ID : %2d  Name : %5s  Sal : %3u  PF : %.2f",
            e.id, e.name, e.sal, e.pf);
   }
   pf(" ____________________ ");
   fclose(old);
   getch();
   return 0;
}
void shaocur()
{
   int ne, i;
   ne= curdata();
   if (ne == 0)
   {
      printf("\n ***** NO RECORDS IN CUR LIST *****");
      getche();
      return;
   }
```

```
    cur = fopen( curfile, "rb");
    if (cur == NULL)
    {
        printf("\n Unable to read cur list ");
        getch();
        return ;
    }
    Printf("\n---------- CUR RECORD LIST ---------- ");
    for(c=1; c<=ne; c++)
    {
        fread(&e, sizeof(EMP), 1, cur); // file data to application
        printf("\n ID :%2d  Name: %5s  Sal: %3u  Pf: %.2f ",
        e.id, e.name, e.sal, e.pf);
    }
    Printf("\n ---------- ");
    fclose(cur);
    getch();
    return ;
}
-> void displayemp()
{
    int option;
    printf("\n CURRENT Records  1 : ");
    printf("\n OLD     Records  2 : ");
    printf("\n ALL     Records  3 : ");
    scanf("%d", &option);
    switch(option)
    {
        case 1 : showcur();
                 break;
        case 2 : showold();
                 break;
        case 3 : showcur();
                 showold();
                 break;
        default : printf("\n INVALID OPTION....");
    }
    getch()
    return ;
}
```

```
int main()
{
    int option;
    void updateemp(); // declaration
    void deleteemp(); // declaration
    while(1)
    {
        clrscr();
        printf("\n FOR CREATE RECORD .... 1 : ");
        printf("\n FOR DISPLAY RECORDS .... 2 : ");
        printf("\n FOR NO. OF CUR RECORDS .. 3 : ");
        printf("\n FOR NO. OF OLD RECORDS .... 4 : ");
        printf("\n FOR MODIFY RECORD- - - - - - - 5 : ");
        Printf("\n FOR DELETE RECORD- - - - - 6 : ");
        printf("\n FOR EXIT . . . . . 7 : ");
        Printf("\n PLEASE ENTER OPTION . . . . : ");
        scanf("%d", &option);
        switch(option)
        {
            case 1 : createemp();
                     break;
            case 2 : displayemp();
                     break;
            case 3 :
        Printf("\n No. of Records in CURRENT DataBase : %d", curdata());
                 getch();
                 break;
            case 4 :
        Printf("\n No. of Records in OLD DataBase : %d", olddata());
                 getch();
                 break;
            case 5 : updateemp();
                     break;
            case 6 : deleteemp();
                     break;
            case 7 : return 0;
```

```
            default: printf("\n Invalid option..");
                     getch();
                     break;
        }
      }
    }
```

1/8/12

```
void updateemp()
{
  int i, id, ne, no1, flag = 0;
  ne = curdata();
  if (ne == 0)
  {
    printf("\n NO DATA TO UPDATE ");
    getch();
    return;
  }
  printf("\n Enter EMP ID to update : ");
  scanf("%d", &id);
  no1 = olddata();
  if(id < 1 || id > ne+no1)
  {
    printf("\n INVALID ID");
    getch();
    return 0;
  }
  cur = fopen(curfile, "r+b");
  if (cur == NULL)
  {
    printf("\n UNABLE TO UPDATE DATABASE");
    getch();
    return;
  }
  for( i=1; i<=ne; i++)
  {
    fread(&e, sizeof(EMP), 1, cur);
    if(e.id == id)
    {
      printf("\n ID : %d  Name : %s  Sal : %u  PF : %.2f",
        e.id, e.name, e.sal, e.pf);
```

```
            fseek ( cur, sizeof (EMP) * (id-1) , SEEK_SET);
            printf ("\n ENTER NEW NAME : ");
            flushall ();
            gets (e.name) ;
            printf ("\n Enter new salary : ");
            scanf (" %u", &e. sal);
            e.pf = (float) e.sal * 10/100 ;
            fwrite (&e, sizeof (EMP) , 1 , cur);
            flag=1;
            break;
        }
    }
    if (flag==1)
    {
        printf ("\n RECORD is UPDATED");
        getch();
        fclose (cur);
        return;
    }
    else
    {
        printf ("\n RECORD NOT FOUND");
        getch();
        fclose (cur);
        return;
    }
}
void deleteemp()
{
    int i, id, ne, ne1, flag=0;
    FILE * temp;
    ne = curdata() ;
    if (ne == 0)
    {
        printf ("\n NO DATA TO DELETE ");
        getch ();
        return ;
    }
    printf ("\n ENTER EMP ID TO DELETE ");
    scanf ("%d", &id);
    ne1 = oldata() ;
```

```
if (id < 1 || id > no+no-1)
{
    printf("\n INVALID NO");
    getche();
    return;
}
cur = fopen(curfile, "rb");
old = fopen(oldfile, "ab");
temp = fopen("E:\\temp.dat", "wb");
for(i=1; i<=no; i++)
{
   fread(&e, sizeof(EMP), 1, cur);
   if(e.id==id)
   {
      fwrite(&e, sizeof(EMP), 1, old);
      flag=1;
   }
   else
      fwrite(&e, sizeof(EMP), 1, temp);
}
fclose(temp);
fclose(cur);
fclose(old);
remove(curfile);
rename("E:\\temp.dat", curfile);
if(flag==0)
{
   printf("\n RECORD IS DELETED");
   getch();
   return;
}
else
{
   printf("\n RECORD NOT FOUND");
   getch();
   return;
}
}
```

conio.h :-

- Console related all predefined functions are declared in conio.h.
- conio.h is compiler dependent header file, that is UNIX or LINUX based compilers don't support conio.h.
- conio.h related predefined functions are :-

cgets	clreol	delline	cprintf
cputs	cscanf	movetext	insline
getpass	gotoxy	wherey	wherex
kbhit	textbackground		textcolor

1) cprintf() :-

- By using cprintf() function, we can print the data on console in color's format.

2) cscanf() :-

- By using cscanf() function, we can read the data from user in colors.

3) gotoxy() :-

- It is a predefined function, which is declared in conio.h, by using this function, we can pass the control any where on screen with the help of (x,y) co-ordinate values.
- gotoxy() function requires two arguments of type int, i.e. x, y cordinate values.
- Syntax :- void gotoxy(int x, int y);

4) wherex() :-

- By using this predefined function, we can find horizontal cursor position, i.e. 'x'-cordinate value.
- wherex() function returns an integer value, i.e. x-axis value.
- Syntax :- int wherex(void);

5) wherey() :-

- By using this predefined func", we can find vertical cursor position, i.e. y'cordinate value.
- wherey() func" returns an integer value, i.e. y-axis value.
- Syntax :- int wherey(void);

⑥ textcolor() :-

- by using this predefined function, we can change the color of the text.
- textcolor() function acquires one argument of type int, i.e. newcolor value
- Syntax: void textcolor(int newcolor);
- textcolor() funcn supports 16 colors from the range of 0 to 15.

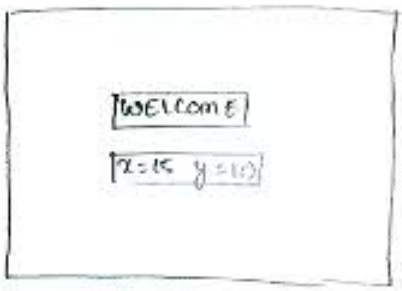
0 → Black (0-15) → Combination
15 → white

⑦ textbackground:-

- by using this predefined funcn, we can change the background color of the text.
- textbackground() funcn acquires an argument of type int, i.e. newcolor value.
- Syntax: void textbackground(int newcolorvalue);

ex1)

```
#include <conio.h>
int main()
{
    int x,y;
    gotoxy(15,8);
    textcolor(9);
    textbackground(15);
   //printf("WELCOME"); (1,1) Point & no color
    cprintf("WELCOME");
    x = wherex();
    y = wherey();
    gotoxy(15,10);
    textcolor(15);
    textbackground(9);
    cprintf("x=%d y:=%d",x,y);
    getch();
    return 0;
}
```



ex2) /* to display the system time & date on console */

```
struct time
{
    unsigned char ti_hour;
    unsigned char ti_min;
    unsigned char ti_sec;
    unsigned char ti_hund;
};
```

```
struct date
{
    int da_year;
    unsigned char da_day;
    unsigned char da_mon;
};
```

note: time & date are two predefined structures in dos.h conio.h

---

DCLOCK.C

```
#include <dos.h>
#include <conio.h>
int main()
{
    struct time t;
    struct date d;
    while (!kbhit())
    {
        getdate(&d);
        gettime(&t);
        gotoxy(15,10);
        textcolor(5);
        textbackground(15);
        cprintf("%2.2d/%2.2d/%d", d.da_day, d.da_mon, d.da_year);
        cprintf("  %2.2d:%2.2d:%2.2d:2.2dd",
                t.ti_hour, t.ti_min, t.ti_sec, t.ti_hund);
    }
}
```

note: kbhit() → Keyboard hit

(note)

↳ struct time is a predefined structure, which is declared in dos.h

↳ By using struct time, we can hold time properties.

↳ Struct time contains 4 variables, i.e.

ti_hour,
ti_min,
ti_sec, &
ti_hund.

↳ (syntax)

```
struct time
{
    unsigned char ti_hour;
    unsigned char ti_min;
    unsigned char ti_sec;
    unsigned char ti_hund;
};
```

→ struct date is a predefined structure, which is defined in dos.h, by using this struct we can manage date properties.

→ The properties of date structure are :- da_year, da_mon, da_day.

→ (def^n)

```
struct date
{
    int da_year;
    char da_day;
    char da_mon;
};
```

gettime() :-

↳ It's a predefined function, declared in dos.h

↳ By using this function, we can load system time.

settime() :-

↳ By using this predefined function, we can change system time.

getdate() :-

↳ By using this predefined function, we can load (find) system date.

↳ getdate() funct^n loads system date information to application variable.

setdate() :-

↳ By using this predefined function, we can change system date.

Scan code :-

- ASCII code is a combination of single integer value, Scancode is a combination of two integer values.

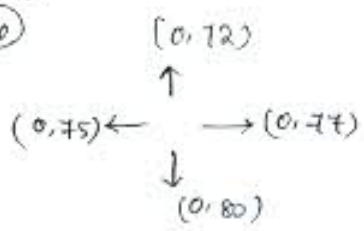
- All functional keys work with the help of scan codes.

ex :- →, ↓, ←, ↑, pagedown, pageup .. etc.

- whenever we are working with scancode, the first value will be '0' always.

ex)
```
#include <stdio.h>
#include <conio.h>
int main()
{
    char ch1, ch2;
    ch1 = getch();
    ch2 = getch();
    printf("\n Scan code :- ch1 : %d  ch2 : %d", ch1, ch2);
    getch();
    return 0;
}
```

ex)



page up → 0 73

page down → 0 81

ex)
```
// arrow.c
#include <conio.h>
int main()
{
    char ch1, ch2;
    int x, y;
    ch1 = getch();
    while( ch1 == 0)
    {
        ch2 = getch();
        x = wherex();
        y = wherey();
        if (ch2 == 77)
            gotoxy (x+1, y);
```

```
        if(ch2 == 75)
            gotoxy(x-1, y);
        if (ch2 == 72)
            gotoxy (x, y-1);
        if (ch2 == 80)
            gotoxy (x, y+1);
        if (ch2 == 73)
            gotoxy (x, 1);
        if (ch2 == 81)
            gotoxy (x, 25);
        ch1 = getch();
    }
}
```

cputs() :-

- By using this predefined function, we can ~~Predict~~ print string data on console in colors format.

cgets() :-

By using this predefined function, we can read the string data from user in colors format.

ex4)

```
#include <conio.h>
#include <dos.h>
int main()
{
    int i, j;
    int x1 = 1, y1 = 1;
    int x2 = 75, y2 = 1;
    int x3 = 1, y3 = 25;
    int x4 = 75, y4 = 25;
    char str[] = "Hello";
    clrscr();
    i = j = 0;
    while(!kbhit())
    {
        textcolor(i+1);
        textbackground(j);
        gotoxy(x1+i+i+i, y1+i);
        cprintf(str);
```

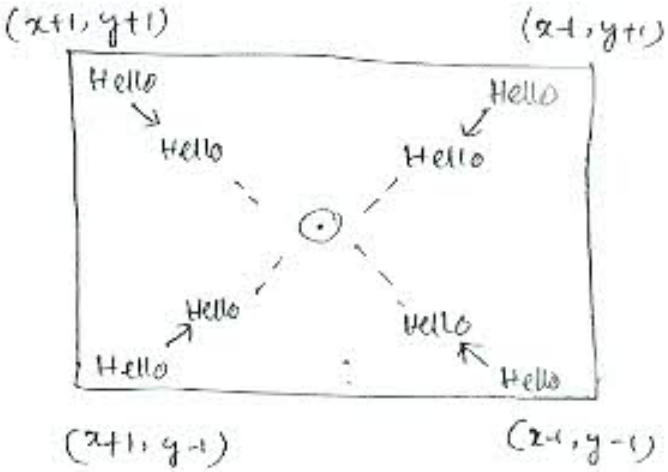
```
        textcolor (j+2);
        textbackground (j+1);
        gotoxy (x2-i-i-i, y2+i);
        cprintf (Str);

        textcolor(j+3);
        textbackground(j+2);
        gotoxy(x3+i+i+i, y3-i);
        cprintf(Str);

        textcolor(j+4);
        textbackground(j+3);
        gotoxy(x4-i-i-i, y4-i);
        cprintf(Str);
        i++;
        j++;

        if(j==9)
            j=0;

        if (i==8)
        {
            sound(200);
            i=0;
            delay(1000);
            nosound();
        }
        delay(500);
        clrscr();
    }
    return 0;
}
```



sound()

- sound() is a predefined function, declared in dos.h, by using sound() function, we can make mother board speaker in 'on' mode.
- by using nosound() function, we can make the mother board speaker in 'off' mode.

insline()

(insert line)

- By using this predefined function, we can insert a row on console

delline()

(delete line)

- By using this predefined function, we can delete a row from console.

ex:> #include <conio.h>

```
void main()
{
    char str[10]="welcome";
    int i;
    for(i=1; i<=10; i++)
    {
        textcolor(i);
        textbackground(i+1);
        gotoxy(10,i);
        cputs(str);
        cprintf(" #%d", i);
    }
    gotoxy(10,7);
    getch();
    delline();
    gotoxy(10,4);
    getch();
    textcolor(15);
    textbackground(0);
    insline();
    getch();
}
```

- clrscr()

→ By using this predefined function, we can clear the data on console.

→ when we are using clrscr, then complete console data will be cleared.

clreol() → (clear end of the line)

→ By using this predefined function, we can clear specific part of the line from console.

ex6)
```
#include<conio.h>
int main()
{
    char str[10] = "welcome Hello";
    gotoxy(10,10);
    textcolor(5);
    textbackground(15);
    cputs(str);
    getch();
    gotoxy(15,10);
    getch();
    clreol();
    getch();
    return 0;
}
```

GRAPHICS :-

- Applying the visual properties to a dos application is called Graphics.
- Whenever we are working with Graphics appl^n, always we require to include "graphics.h"
- "graphics.h" provides all the predefined function forward declaration.
- Whenever we are working with graphics related application, then we require to find out "EGAVGA.BGI" file location.
- The above file will install all the resources of graphics to the application.
- Generally, the file is available in "C:\TC\BGI" directory.
- Whenever we are working with any graphics appl^n, then we require to initialize graphics properties properly.
- By using "initgraph()" function we can initialize graphics property.
- At the time of initializing the graphics due to graphics resources, we can get initialization related errors also.
- Initialization related errors can be found by using "graphresult()" function
- 'graphresult()' function returns error code, if the graphics are not initialized properly, if it is initialized properly, then we will get "grOK".

- By using "graph errormsg()" funct^n, we can display error message
- By using "closegraph()" funct^n, we can close all properties of the graphics.

— o —

ex1) /*

```
#include <conio.h>
int main()
{
    char far * ptr = (char far *) 0xB8000000;
    clrscr();
    *(ptr+0)='B';
    *(ptr+1) = 3;
    *(ptr+2) = 'A';
    *(ptr+3) = 4;
    *(ptr+4) = 'P';
    *(ptr+5) = 5
    *(ptr+6) = 'O';
    getch();
    return 0;
}
```

—: End of C' :—

Bhai 01/12/12

01/12/12