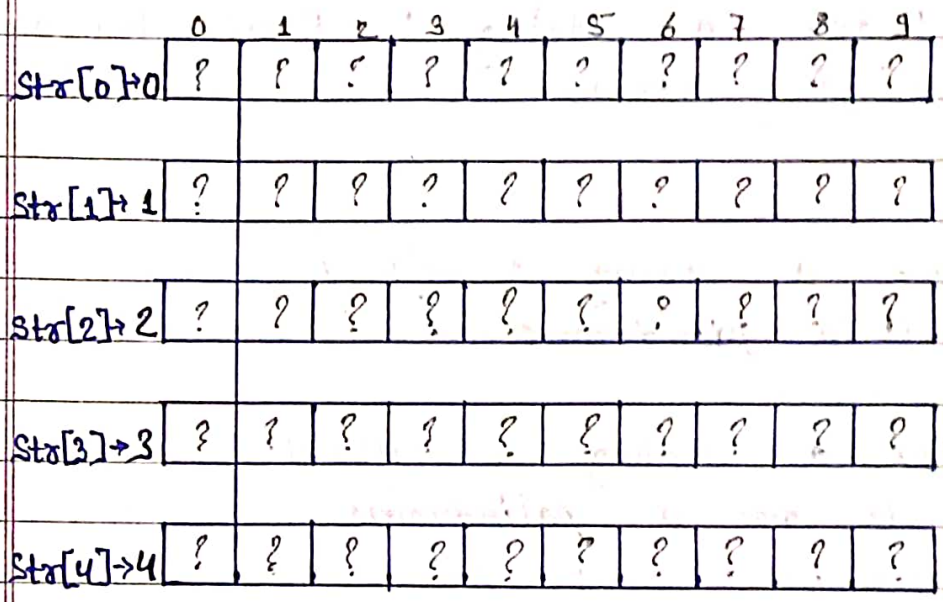
Double Dimensional character Array.

⇒ Whenever we want to store multiple strings In our program then in C language we prefer to use a Double Dimensional character Array.

Syntax :-

```
char <arr_name>[<row_size>][<column_size>];
```

Ex :- `char str[5][10];`



Example :-

```
int main()
{
  char str[5][10];
  int i;
  for(i=0; i<5; i++)
  {
```

```
        printf("Enter name : ");
        scanf("%s", str[i]);
    }
    for (i=0; i<5; i++)
        printf("\n%s", str[i]);

    return 0;
}
```

		0	1	2	3	4	5	6	7	8	9
str[0] →	0	'A'	'J'	'A'	'Y'	'\0'	?	?	?	?	?
str[1] →	1	'R'	'a'	'h'	'u'	'l'	'\0'	?	?	?	?
str[2] →	2	'G'	'e'	'e'	't'	'a'	'\0'	?	?	?	?
str[3] →	3	'V'	'i'	'n'	'e'	'e'	't'	'\0'	?	?	?
str[4] →	4	'V'	'i'	'k'	'a'	's'	'h'	'\0'	?	?	?

str

Q5 WAP to accept 5 strings from the user, store it in a Two Dimensional character array and print the length of each name using the appropriate string function.

Solution:-

```
int main()
{
  char str[5][10];
  int i,x;
  for(i=0; i<5; i++)
  {
    printf("Enter name: ");
    scanf("%s", str[i]);
  }
  for(i=0; i<5; i++)
  {
      x = strlen(str[i]);
      printf("\n Length = %d", x);
  }
  return 0;
}
```

(i) Rewrite the previous program but without using strlen()

(ii) Rewrite the previous program but without using x variable.

(i)

Solution:-

```
int main()
{
  char str[5][10];
  int i, j;
  for( i=0; i<5; i++)
  {
```

very important

```
        printf("Enter  name ");
        scanf("%s", str[i]);
    }
    for (i=0; i<5; i++)
    {
        for (j=0 ; str[i][j] != '\0' ; j++ );
        {
            printf("\n Length is : %d ", j);
        }
    }
    return  0;
}
```

Q) WAP to accept 5 names from the user, store it in a Two Dimensional character Array and print the reverse of each name.

FUNCTIONS

Code body

```
Void main ()
{
    . . . . . .
    - - - - - -
    - - - - - -
    # # # #
    # # # #
    # # # #
    # # # #
    - - - - - - -
    - - - - - -
    - - - - - -
    - - - - - -
    - - - - - -
    # # # #
    # # # #
    # # # #
    # # # #
    - - - - - -
    - - - - - -
    - - - - - -
    - - - - - -
    - - - - - -
    # # # #
    # # # #
    # # # #
    # # # #
    - - - - - -
    - - - - - -
    - - - - - -
    - - - - - -
    - - - - - -
}
```

→ code of average

→ code of average

→ Code of average.

Solution:-

```
int main()
{
    - - = = =
    - - - - -
    - - - - -
    - - - - -
    - - - - -
    average(); → fun^n call
    - - - - -
    - - - - -
    - - - - -
    - - - - -
    average(); → fun^n call
    - - - - -
    - - - - -
    - - - - -
    - - - - -
    average(); → fun^n call
    - - - - -
    - - - - -
    - - - - -
}
```

function

```
void average()
{
    # # # #
    # # # #
    # # # #
    # # # #
    # # # #
    # # # #
}
```

★ What is function ?

- Function are small blocks of code, having a particular name, followed by a pair of pranthesis, designed to perform a particular task.

Date
Page

- These function can be called by the main program whenever the need of the task performed by a function arises.

★ what are the advantages of using a function ?

1) Functions help us reduce the length of the code. This is because we can write the repetitive code inside a function and using just a single line, we can call the function. This saves the efforts of the programmer as well as helps us reduce the length of the main code.

2) By using functions, we can easily detect and rectify errors in our program because we get a fair idea regarding the error point as well as and correction done inside the function body automatically reflects at every place where the function has been called.

3) By learning how to create functions we also can learn to create our own header files. Due to this we can keep all our functions in one header file and use this header file in other programs also which will allow us to use these functions also.

For Example :-

we can create a header file called as mymath.h containing our own functions like factorial(), prime(), evenodd() etc. and in every program where we include this header file, there we can use these functions also.

So we can say that functions provide code reusability.

☆ Types of Functions :-

i) Pre-defined function
ii) User defined function

	Pre-defined	User-defined function
⇒	which are already present in compiler.	which a programmer creates for himself.
Ex:-	printf(), getch(), fflush(), clrscr(). etc.	factorial(), prime(), Armstrong(). etc.

★ Steps required for developing a function:

1) Function Declaration :-

Telling something about the function to the compiler, before using it is called as function declaration.

2) function call :-

Invoke a function, so that it starts running is called function call.

3) Function Definition or Function Body :-

writing a set of statements inside the curly brace of functions to perform the actual task of the function.

* Important points :-

main() is defined by programmer.

main() is called by the Operating System.

main() is declared inside the compiler.

Example :-

i) clrscr() is declared inside header file conio.h.

ii) clrscr() is called by programmer

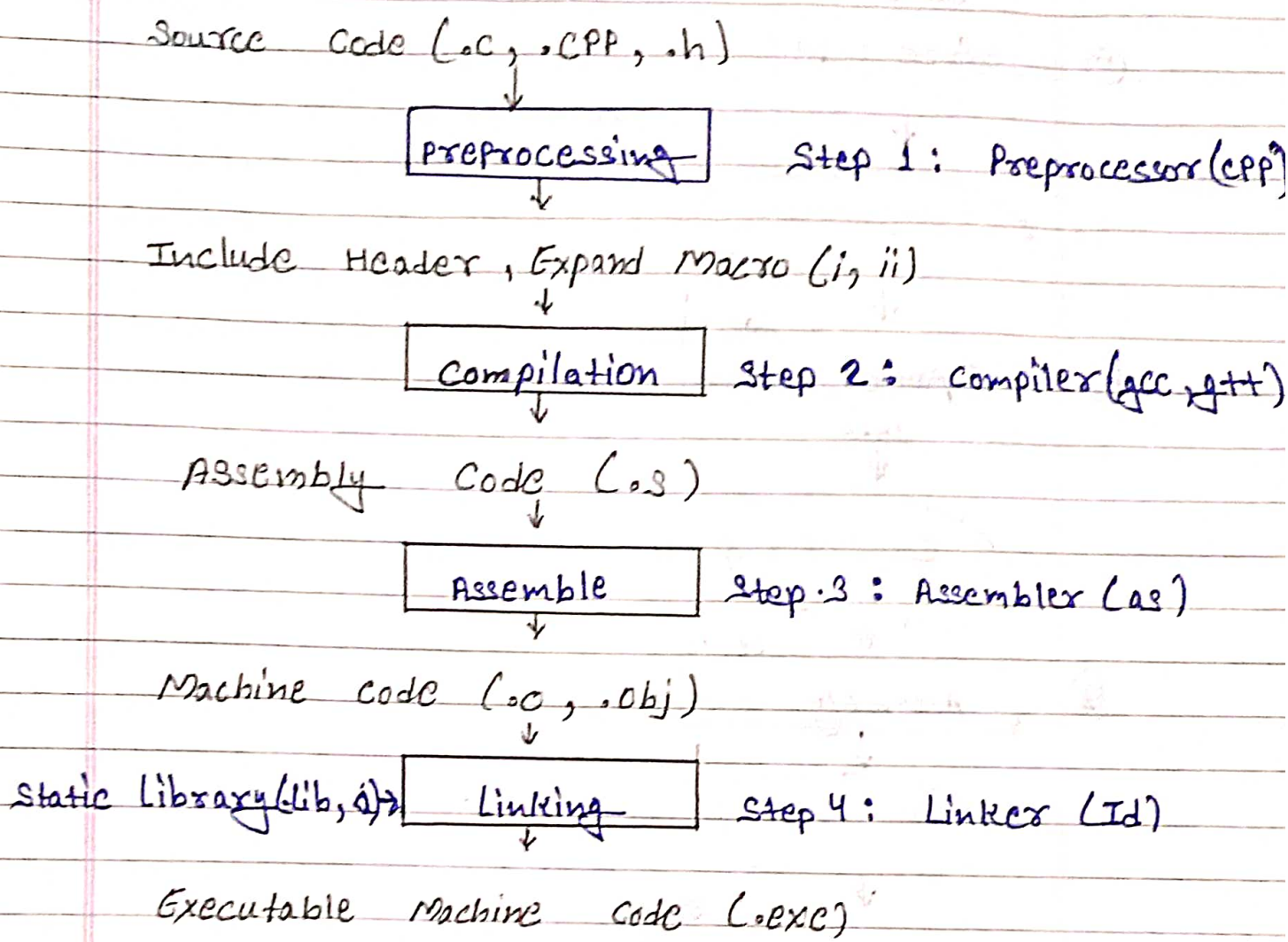
iii) clrscr() is defined in library file.

Date
Page

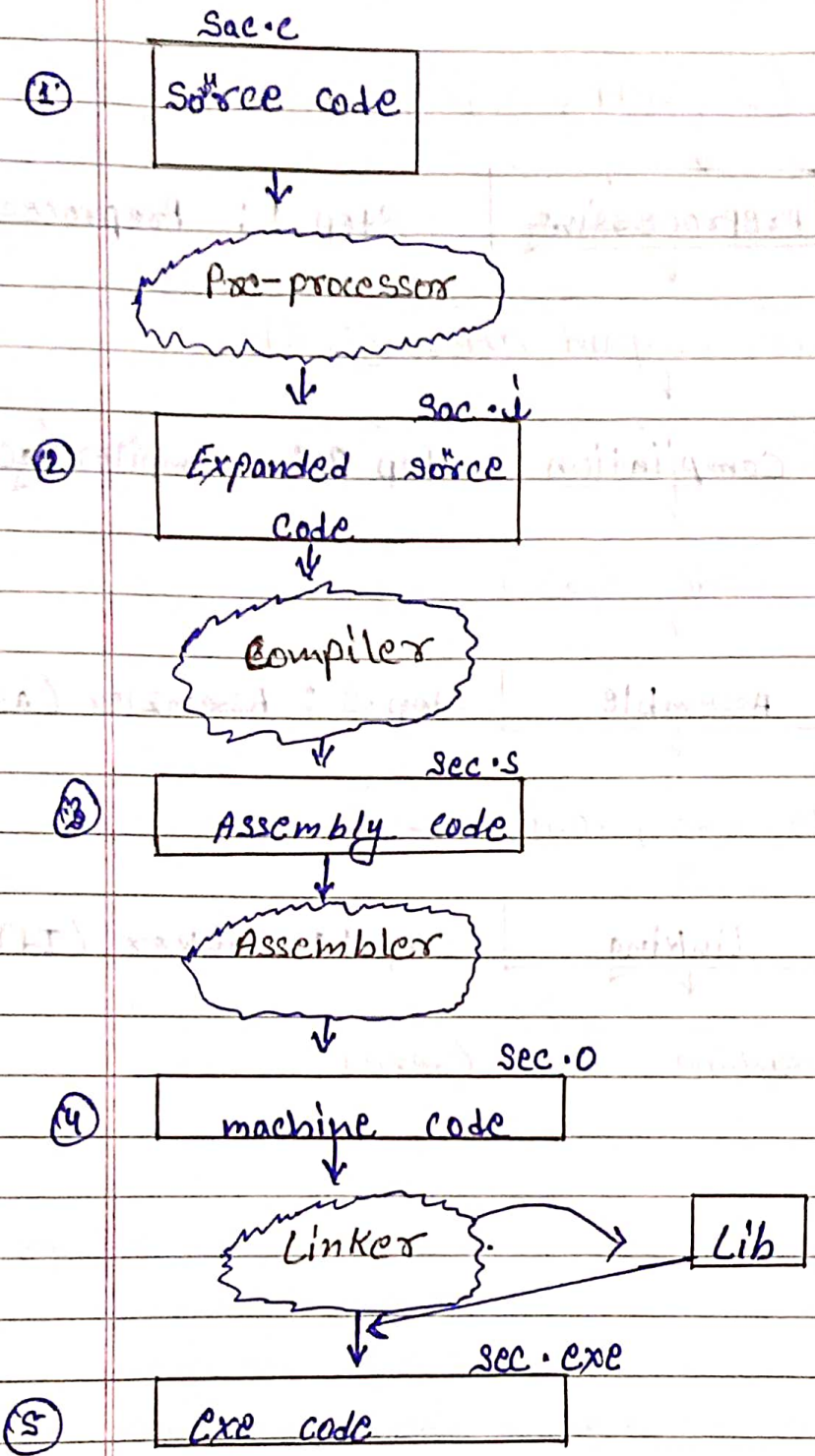
☆ Header Files V/S Library files.

	Header File	Library Files
1)	Header files contain function declarations.	Library files contain function definition
2)	They have the extension ".h".	They have the extension ".lib".
3)	They are located inside a folder called INCLUDE which is itself inside a folder called TC or TURBOC3 etc.	They are located inside a folder called LIB which is itself inside a folder called TC or TURBOC3 etc.
4)	They are text files and thus can be easily read by anyone.	They are machine code or binary codes and thus are not understand-able to humans.
5)	They are added in the program by the programmer using the statements #include which is handled by special software called as pre-processor.	They are included in our program by a special software called as linker.

★ LIFE CYCLE OF A C program



For Better Understanding



"LIFE CYCLE of C Program"

" Difference b/w header files & Library file "

	Header File	Library file
when gram ↓	They are attached to our program we write #include Statement and the Preprocessor attaches the complete code of the header file in our program.	These files are attached to our program by another very special software called as LINKER.

★ Components of Function Declaration :—

1) Return type
2) Function Name
3) Function Arguments , Function Parameters

★ Syntax of function declaration or Prototype

<return_type><function_name>(<arguments>);

The value which the function will return

Any meaningful name (Rules applicable for variable)

Examples :-

			header files
i)	void clrscr(void);	→	conio.h
ii)	double pow(double, double);	→	math.h
iii)	int strcmp(char[], char[]);	→	string.h
iv)	unsigned int strlen(char[]);	→	string.h
(v)	double sqrt(double);	→	math.h
(vi)	float average(int, int, int);	→	user defined function

Qs) Why do we write void or int before main() ?

Ans :- Since, main() is a function, it is bound to have a return type. Now as we defined main(), it is our responsibility to mention the return type of main() while writing it's body. So we write either "void" or "int" as the return of main()

Q) what is difference b/w void main and int main ?

Ans:- The meaning of the "void" in front of main(), we as a programmer do not want to return any value from main() to the OS, which is the caller of the function main().

But it is not considered to be a good programming practice if we write "void" main().

Rather programmers are advised to use "int" as the return type of main().

If the return type of main() is int, then programmer has to return either of the 2 values from main() and these values are :

1) 0 : This indicates successful termination.

☆ Components of function Declaration :-

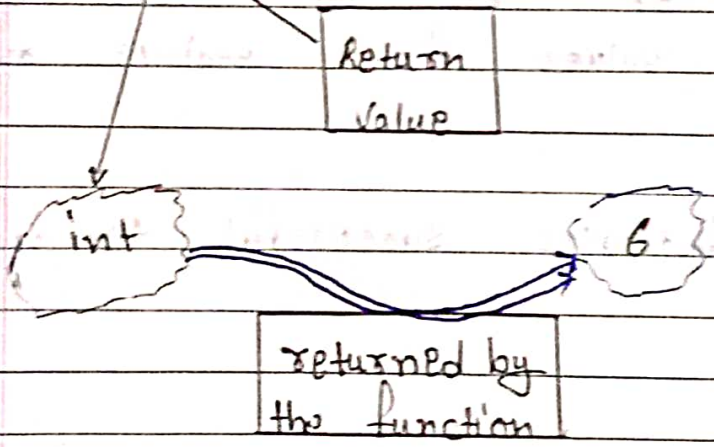
1) Return type
2) Function Name
3) Function Arguments , Function parameters

Ex :-

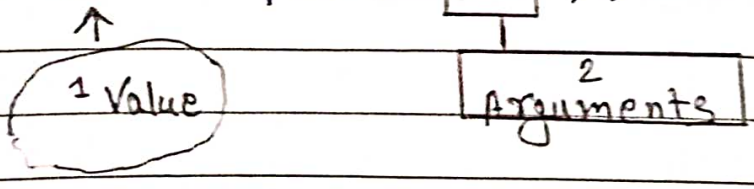
i) `x = pow( 3, 2 );`

Function Arguments

ii) `x = strlen("Sachin");`



iii) `x = pow( 3, 2 );`



iv) `x = strcmp ("Ajay", "Ajit");`

☆ Using function delay() :-

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

void main()
{
    int i;
    for(i=1; i<=10; i++)
        {
            printf("%d \n", i);
            delay(1000);
        }
getch();
}
```

→ No return value

↳ Argument (in milliseconds)

Date
Page

★ Using function sleep() :-

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

Void main()
{
    int i;
    for(i=0 ; i<=10 ; i++)
      {
          printf("%d\n", i);
          sleep(1); → No return value
      }                ↳ Argument (in seconds).
  getch();
}
```

★ Using function textcolor() & gotoxy() :-

```
#include<conio.h>
#include<stdio.h>
Void main()
{
    clrscr();
    textcolor(YELLOW);
    gotoxy(40, 12);
    cprintf("Hello\n");          row → 40 12
    getch();                     column → 40
}
```

☆ Echo and buffered :-

1)
```
Void main()
{
    clrscr();
    char ch;
    printf(" Enter a character : ");
    scanf(" %c", &ch); // echo or buffered
    printf("\n You entered %c \n", ch);
    getch();
}
```

Output :-

Enter a character : a ← buffered (After pressing Enter)

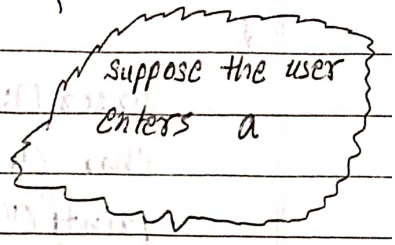
You entered a

दबाया (echo)

2)
```
#include <stdio.h>
#include <conio.h>
Void main()
{
    clrscr();
    char ch;
    printf(" Enter a character ");
    ch = getch();
    printf("\n You entered %c \n", ch);
    getch();
}
```

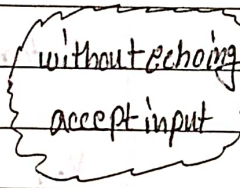
Output :-

Enter a character a
You entered a



3)

```
#include <stdio.h>
#include <conioh>
void main()
{
    clrscr();
    char ch;
    printf("Enter a character : ");
    ch = getch();
    printf("*");
    printf("\n You entered %c\n", ch);
    getch();
}
```



Output :-

Enter a character : *
You entered a

Note:- SANSITIV INPUT

i) Password
ii) OTP
iii) CVV

Date
Page

Q) write a program to ask the user to input password. The password will be of 4 character. On every keystroke an * should be desplayed on the screen. If the password is correct then the massage PASSWORD ACCEPTED should be desplayed otherwise code should display the message INVALID PASSWORD.

Assume that correct password is "abcd"

Sample output

```
Enter Password : ****
PASSWORD ACCEPTED
Enter Password : ****
INVALID PASSWORD
```

Solution:-

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

Void main()
{
 char pwd[5];
 int i;
 printf(" Enter password : ");
 for(i=0 ; i<4 ; i++)
 {
     pwd[j] = getch();
     printf(" * ");
```

```
}
pwd[i] = '\0';

if(strcmp(pwd, "abcd") == 0)
    printf("\n Password Accepted");
else
    printf("\n Invalid Password");

getch();
}
```

Above program using delay() :-

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

void main()
{
    clrscr();
    char pwd[5];
    int i;
    printf("Enter Password : ");
    for(i=0; i<4; i++)
    {
        pwd[i] = getch();
        printf("*");
    }
    pwd[i]= '\0';
    printf("\n Password Verification under process
```

```
please wait.... ");

delay (3000);
if ( strcmp (pwd , "abcd" ) == 0)
    printf(" \n Password Accepted ");
else
    printf (" \n Invalid password ");

getch ();
}
```

Q) Modify the previous program so that if the password is wrong, then provide two more attempts to the user so that he can try again.

Q) Modify the previous program and allow the user to press BACKSPACE key and delete the previous character

Q) Modify the previous program so that as soon as the user types a character, display it for 1 sec. and then replace it with *.

Q) What is the return type of the function printf() ?

Ans :—

Q) Is printf() a function ?

Ans :— Yes printf() is a function.

Q) Can You explain what the following code is doing ?

```
printf("Hello");
```

Ans :— This code is performing 2 things.

a) it is printing "Hello" on console.
b) it is also returning 5.

This is because the function printf() returns the length of the printed data

Ex :—

```
int x, y, z;
x = printf("I Love INDIA");
y = printf("%d", x);
z = printf("%d", y);
printf("%d", z);
```

Output :— I Love INDIA12 21

Q5 What does scanf returns ?

Ans:- It returns total numbers of inputs scanned.

Ex:-
```
int age, x, y;
char name[20];
printf("Enter Your name and age : ");
x = scanf("%s %d", name, &age);
y = printf("%s %d", name, age);
printf("\n x = %d, y = %d", x, y);
```

Output:-

```
Enter Your name and age : Ravi 35
Ravi 35
x = 2, y = 7
```

TYPES of Functions

Based on discussion we can say that in C language, functions are of 4 types.

1) Takes Something and Returns Something

strlen(), pow(), sqrt(), strcmp() etc.

2) Takes Something and Returns Nothing :

fflush(), delay() , textcolor(), gotoxy(); etc.

3) Takes Nothing and Returns Something :-

getch().

4) Takes Nothing and Returns Nothing :

clrscr().

A function to calculate average of 3 integers using first style of function designing.

Solution :-

```
#include<stdio.h>
float average(int, int, int);   <- function declaration or function prototype
int main()
{
    int a, b, c;
    float d;
    printf("Enter 3 integers : ");
    scanf("%d %d %d", &a, &b, &c);
    d = average(a, b, c);        <- function call
    printf("Average is %f", d);
    return 0;
}
```

```
float average (int i , int j , int K)
{
    float x ;
    x = (float)(i + j + K) / 3 ;
    return @ x ;
}
```

function or function body.

Qs what is return ?

Ans :-

1) return is a keyword .

2) we use return for returning back a value to the caller function .

3) Syntax :

```
return (<value>);
```

or

```
return value ;
```

Example of return :-

```
1) return (10);
2) return (q);
3) return (q + r);
4) return (p , q);
        or
5) return p , q ;
```

or

```
6) return p ;
   return @ q ;
```

" Exercise "

Q→ WAP to create a function called factorial() which should accept an integer as argument and should calculate and return its factorial. Make sure that your function return 1 if 0 is passed as argument.

Q→ WAP to create a function called convert() which should accept a character as argument and return it by converting from lower case to upper case and vice versa. If the passed character is not an alphabet then it should be returned as it is.

★ Calculating factorial using First style of Designing.

Solution:—

```
#include<stdio.h>
int fact(int);
int main()
{
    int n, f;
    printf("Enter a no:");
    scanf("%d", &n);
    f = fact(n);
    printf("Fact is %d", f);
    return 0;
```

```
}
int fact(int n)
{
    int f = 1;
    while(n>1)
    {
        f = f * n;
        n--;
    }
    return f;
}
```

Calculating Average of 3 integers using second style of Function Definition :-

```
#include <stdio.h>
void average(int, int, int)
void main()
{
    int a, b, c;
    printf(" Enter 3 int : ");
    scanf(" %d %d %d", &a &b &c);
    average(a,b,c);
    return 0;
}
void average(int i, int j, int k)
{
    float x;
    x = float(i+j+k)/3;
    printf(Average is %f ", x);
    return 0;
```

Q5 WAP to create a function called show table() . This function should accept two arguments representing the number and number of terms and it should print the table of first number up to the numbers terms given by the second number .

Sample output :-

```
Enter a number : 7
Enter number of terms : 3
7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
```

Solution :-

```
#include <stdio.h>
void average (int,int,int)
int main()
{
    int a, b, c;
    printf(" Enter 3 int ");
    scanf(" %d %d %d ", &a, &b, &c);
    average (a, b, c);
    return 0;
}
void average (int i, int j, int k)
{
    float x;
    x = (float) (i+j+k)/3;
    printf(" Average is %f , x);
    return ;
```

Date
Page

Etc. Calculating Average using Third Style

we use this style when we have unknown arguments :-

```
#include <stdio.h>
float average (void);
int main()
{
    float d;
    d = average();
    printf("Average is %f.",d);
    return 0;
}
float average()
{
   int a,b,c;
   float d;
   printf("Enter 3 integers : ");
   scanf("%d %d %d", &a, &b, &c);
   d = (float) (a+b+c)/3;
   return 0;
}
```

⇒ Modify the previous code so that now the function average() can calculate the average of n integers where n is given by the user.

Solution 2—

```
#include <stdio.h>
float average (void);

int main()
{
   float d;
   d = average();
   printf(" Average is %f ", d);
   return 0;
}
float average
{
 int n, i, a sum=0;
 float d;
 printf(" Average of how many numbers ? ");
 scanf("%d , &n);

 for (i=1 ; i<=n; i++)
  {
        printf(" Enter number ");
        scanf(" %d ", &a);
        sum = sum + a;
 }
 d = (float) sum/n;
 return d;
}
```

★ Calculating Average using Forth Style :—

we use this style when no of argument is unknown and no of results unknown.

```
#include <stdio.h>
float average (void);
int main()
{
    float d;
    clrscr();
    d= average();
        return 0;
}
void average (void)
{
    int a, b, c;
    float x;
    printf(" Enter 3 int ");
    scanf(" %d %d %d ", &a, &b, &c);
    x = (float)(a+b+c)/3;
    printf(" Avg is %f ", x);
    return ;
}
```

// Pointer //

Predict the scenarios happens on RAM on the execution of the following lines of codes

```
int main()
{
    int a = 10;
    printf("Value of a is %d", a);
    printf("\n Address of a is %u", &a);
    return 0;
}
```

As we have discussed earlier the address of the variable in Turbo ranges from (0 to 65535) so we use %u

Output:-

10

a → is 1000 ← Assume address of variable a is 1000

Now suppose we have to store the address of variable a into another variable let say variable b

```
int main()
{
    int a = 10;
    int b;
    unsigned int c;
    b = a;
    c = &a;
    return 0;
}
```

we use unsigned int to store address who ranges from 0 to 65535

why it is still giving ERROR

Date
Page

Reason behind this behavior :-

- Although addresses are looks like as normal numbers but for compiler they have special meaning..

- Because a normal variable can hold only value, it can not holds the addresses of the other variables.

- So to store these special numbers we have to declare a special variable which are able to store addresses of other variable in it and for this purpose the topic which we are studying will be used which is the pointer.

What is a pointer ?

A pointer in c language is a VERY SPECIAL variable that can store ADDRESS of other variables.

for Application of the pointer please wait for the end of the topic As the topics goes on you came to know why the pointer is so important in c and why it is also known as the backbone of the c language

* → Value add

& → address of

How do we declare a pointer ?

Syntax :- <data type> *<pointer_Variable_name> ;

★ Important Points of Pointers :-

The data type of a pointer is always decided by looking at the data type of the variable with address the compiler / the programmer will store inside the pointer.

So if a variable is of type int, then the pointer will also be of type int, if the variable is of type char, then the pointer will also be of type char.

```
int main()
{
    int a = 10;
    int *p;
    p = &a;
    printf("Address of a is %u", &a);
    printf("In contents of p is %u", p);
    return 0;
}
```

1000

1000

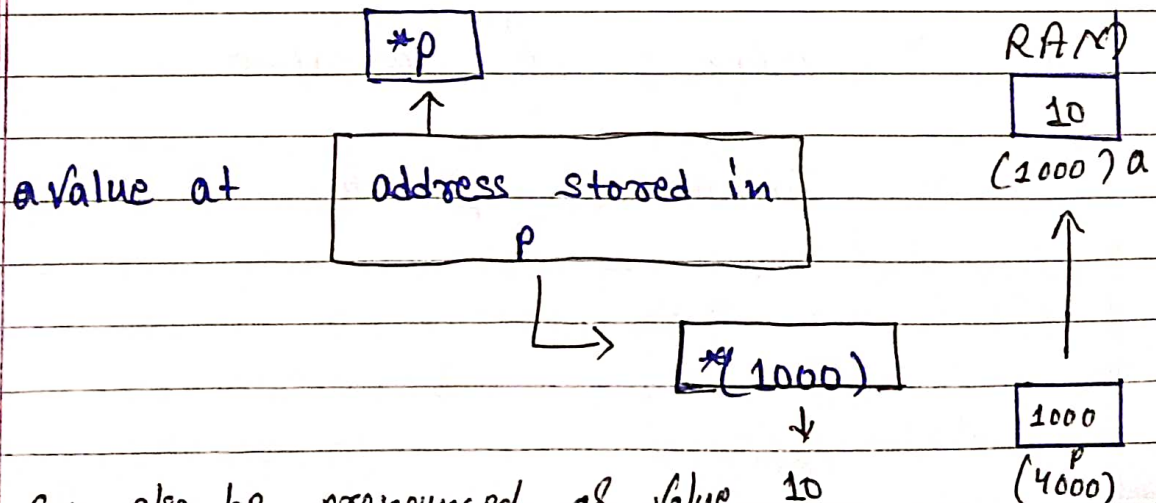
Type of the pointer should be the same as the type of the variable.

The size of the pointer does not depend on the type the size of the pointer is always 2 Bytes in Turbo and is off 4 Bytes for GCC compiler.

```
int main()
{
   int a = 10;
   int *p;
   p = &a;                          → 1000
   printf("Address of a is %u", &a);
   printf("\n contents of p is %u", p); → 1000
   printf("\n %d %d", a, *p);
   return 0;                        → 10 10
}
```

→ Pronunciation of *p

1) Value at the address stored in p



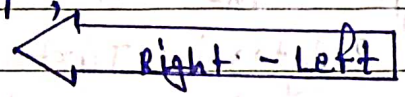
Can also be pronounced as value at p

Date
Page

★ Pronunciation of int *p;

p is a pointer to an integer

int *p;



Do not pronounce it as

integer star p ✗

★ Remember :-

□ *p can't be pronounced as

- Star p ✗
- Asterisk p ✗
- Pointer p ✗

➡ & → Address of operator

* → Indirection operator

Date
Page

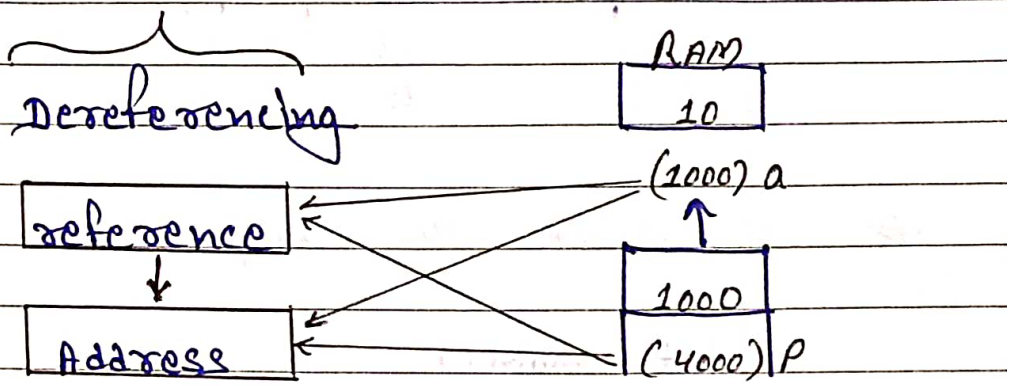
10 ↑ → Value at address stored in p

```
printf (" %d ", *p);
```

Dereferencing Indirect Addressing

Dereferencing :-

Fetching Variable value using its reference (address)



Referencing / Dereferencing :-

because we are assigning address of a in pointer p

```
int main ()
{
    int a = 10;
    int *p;
    p = &a;        → Referencing
    printf(" Address of a is %u ", &a);
    printf("\n contents of p is %u ", p);
    printf("\n %d %d, a, *p);
    return 0;
}
```

↳ Dereferencing of a pointer

Because we are Fetching value of variable a using pointer p

⇒ Direct / Indirect Addressing :-

printf(" %d ", a); → Direct Addressing
↓
10

printf(" %d ", *p); → Indirect Addressing
↓
10

★ Predict the output of the following code

```
int main()
{
    int a = 10;
    int *p;
    p = &a;
    printf(" Address of a is %u ", &a);
    printf("\n Contents of p is %u ", p);
    printf("\n %d %d", a, *p);
    a = 20;
    printf("\n %d %d ", a, *p);
    return 0;
}
```

↳ 20 ↳ 20

```
int main()
{
    int a = 10;
    int *p;
    p = &a;
    printf("Address of a is %u", &a);
    printf("\nContents of p is %u", p);
    printf("\n%d %d", a, *p);

    a = 20;                          -> 20
    printf("\n%d %d", a, *p);
                                     -> 20
    *p = 30;
    printf("\n%d %d", a, *p);
    return 0;                        -> 30  -> 30
}
```

using pointer we can not only access a variable's value but we also can change / modify the variable's value.

☆ Important Points About Pointer :-

→ Statements Prohibited in Pointers :-

1)
```
int a = 10;
int p;
p = &a;        X ERROR
```

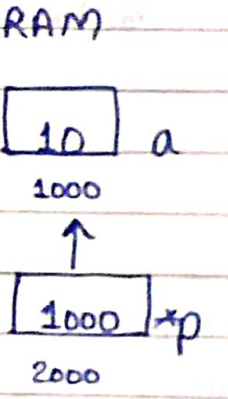
A variable can just hold a value we can never alter variable to hold other variables address.

2)
```
int a = 10;
char *p;
p = &a;   X ERROR
```

The data type of variable and pointer should always same.

☆ How pointer access data from RAM.

```
int a = 10;
int *p;
p = &a;
printf("%d", *p);
```



Pointer only knows starting address of the variable and how many bytes should be picked is decided by the looking at the data type of the pointer which is mentioned during declaration of pointer.

→ Will this code compile and run ?

```
int a = 10;
char *p;
p = &a;
```

i) In C language. (program extension ".c"

* Compile ✓

* Run ✓

ii) In C++ language (Program extension ".cpp")

* Compile ✗

* Run ✗

→ In C language :-

1)

```
int a = 10;
char *p;
p = &a;
printf("%d", *p);
```

output will be correct because binary of first byte will comprise of integer value 10.

2) int a = 256;
char *p;
p = &a;
printf("%d", *p);

output will be 0 Because 256 in binary is of 9 bits value which is beyond of char data type.

⇒ Statements Prohibited in Pointers

3) int a = 10;
int *p;
p = a; ✗ ERROR

we can't assign a value to a pointer

4) int a = 10;
int *p;
*p = a; ✗ (WILD pointer)

we should never dereference a pointer without proper intialization of pointer.

5-)

```
int a = 10;
int *p;   → p is a pointer to an integer
int *q;   → q is a pointer to an integer.
p = &a;   OK ✓
q = &p;   ERROR ✗
```

just like variable can't hold the address of another variable, a pointer also can't hold another pointer's address.

★ Who can hold the address of a pointer:-

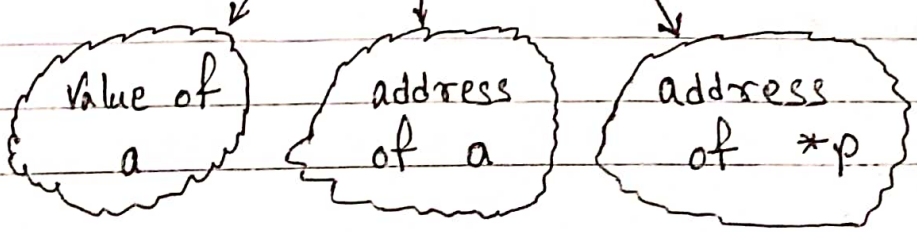
The address of a pointer can only be stored in a POINTER TO A POINTER

```
int a = 10;
int *p;    → p is a pointer to an integer
int **q;   → q is a pointer to an integer pointer
```

```
p = &a;   ok ✓
q = &p;   ok ✓
```

```
printf(" %d %d %d ", a, *p, **q);
```

output :- 10 1000 2000



Date
Page

Using "sizeof" operator / keyword

- "sizeof" is a keyword as well as operator using which we can calculate the size of variables, arrays, pointers, constants, etc.

Syntax :-

1) sizeof (<variable_name>)
   or
2) sizeof (<data_type>)
   or
3) sizeof (<constant>)
   or
4) sizeof (<expression>)

→ unsigned int (Bytes)

→ "sizeof" always returns the value of unsigned int and the size returned by it always in bytes.

Examples :-

1)
```
int a;
char b;
float c;
```

Date
Page

```
printf("%u", sizeof(a));
printf("%u", sizeof(b));
printf("\n%u", sizeof(c));
```

Turbo	GCC
2	4
1	1
4	4

2)

```
printf("%u", sizeof(int));
printf("%u", sizeof(char));
printf("\n%u", sizeof(float));
```

2	4
1	1
4	4

3)

```
int   *p;
char  *q;
float *r;

printf("%u", sizeof(p));
printf("%u", sizeof(q));
printf("%u", sizeof(r));
```

2	4
2	4
2	4

4)

```
printf("%u", sizeof(int*));
printf("\n%u", sizeof(char*));
printf("\n%u", sizeof(float*));
```

2	4
2	4
2	4

Exercise :-

1) printf("\n%u", sizeof(10)); → 2 bytes (int)

2) printf("\n%u", sizeof('A')); → 1 byte (char)

3) printf("\n%u", sizeof("A")); → 2 bytes (string)

4) printf("\n%u", sizeof("Amit")); → 5 bytes (string)

5) printf("\n%u", sizeof(1.5)); → 8 bytes (Double)

6) printf("\n%u", sizeof(1.5f)); 4 bytes (float)

7) int arr[10];

i) printf("\n%u", sizeof(arr)); → 20 bytes

8) char city[10] = {"Bhopal"};

i) printf("\n%u", strlen(city)); → 6 bytes

ii) printf("\n%u", sizeof(city)); → 10 bytes

A Accessing Array using Pointer :-

when we finished the Array chapter, then why are we studying the accessing array using pointer :-

There is a problem with the array which we have studied.

The problem is that if we are using Turbo compiler then the size of the array taken must be a constant.

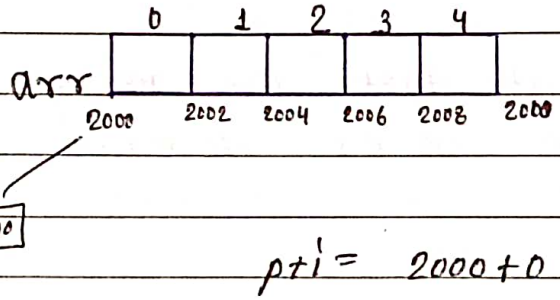
Then either it becomes too ~~too~~ big for the user's needs and so memory will be wasted of either it will be as much small that the code might be crash.

So if we want that the array will be created on the user's demand size then we :-

- we have to access this array by a pointer take input from the user and initialize the array and again print back the value of array on the console window.

First way :-

```
void main()
{
int arr[5], i, *p;
p = arr;
clrscr();
for( i=0; i<=4; i++)
{
    printf(" Enter value");
    scanf("%d", p+i);
}
for( i=0; i<=4; i++)
   printf("%d\n", *(p+i));
```



p+i = 2000+0 = 2000

p+i = 2000+1 = 2002

why this happened?

→ Reason behind this behavior

• Because when we add something in the address then the compiler doesn't solve it like normal math instead of this, it applies a special formula which is

$p + i = p + i * sizeof(<data\_type\_of\_pointer>)$

2000 + 1 * sizeof(int)
2000 + 1 * 2
2000 + 2 = 2002

$p + i = p + i * sizeof(<data\_type\_of\_pointer>)$

2000 + 2 * size of(int)
2000 + 2 * 2
2000 + 4 = 2004

★ Pointer Arithmetic :—

In c programming maths is of 2 types

- Maths of values (Real world Maths)
- Maths of address (Different)

- whenever we will add some value to an address, then the compiler will not directy perfom the addition Rather the compiler multiplies the added value with the sizeof the data type of the address (pointer).

Date
Page

★ what we have learn :-

1) if we add 1 in an integer pointer then 2 Bytes will be added.

2) if we add 1 in a character pointer then 1 Byte will be added.

3) if we add 1 in a float pointer then 4 Bytes will be added.

This also applies on direct addresses :-

Ex :-
```
int a;
printf("%u", &a);      // 5000
printf("\n%u", &a+1);  // 5002
```

★ Accessing an integer array using pointer.

Second way :-

```
void main()
{
 int arr[5], i, *p;
 p = arr;
 for(i=0; i<=4; i++)
 {
    printf("Enter a value :");
    scanf("%d", p);
    p++;
 }
```

```
p=arr;
for(i=0; i<=4; i++)
{
    printf("%d\n", *p);
    p++;                          p++ = p+1
  }                                   2000+1
}                                     2000+2
```

Predict the behavior of the following program:-

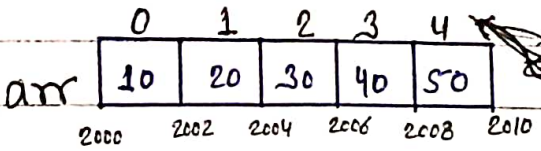
```
void main()
{
    int arr[5] = {10, 20, 30, 40, 50};
    printf("%d", arr[3]);  ← will show 40
    printf("\n%d", arr[10]);
```

why this compiles when we are providing wrong index ?

★ How Compiler solves array expression :-

```
void main()
{
    int arr[5] = {10, 20, 30, 40, 50};
    printf("%d", arr[3]);
    printf("\n%d", arr[10]);
}
```

This is not index



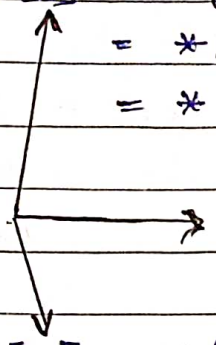
arr[3] = arr + i = arr + i * sizeof(data-type of array)

arr[i] = *(arr + i)

arr[3] = *(arr + 3)

= *(2000 + 3)

= *(2006)



This is not index, it is the offset (distance) to be covered from the base address

arr[10] = *(arr + 10)

= *(2000 + 10)

= *(2020)

→ will compiler also solves this ?

1) printf("%d", 3[arr]);

v/s

printf("%d", arr[3]); → Same

2) printf("%d", arr[-1]);

arr[-1]

⇒ *(arr + (-1))

⇒ *(2000 + (-1))

⇒ *(2000 - 2)

⇒ *(1998)

Date
Page

★ How many ways we have to access / print the array data.

```
Void main()
{
 int arr[5] = {10, 20, 30, 40, 50};
 int i, *p;
 p = arr;
 for (i=0; i<5; i++)
      printf("%d\n", ?);
}
```

1) arr[i]
2) i[arr]
3) *(arr+i)
4) *(arr+i)
5) arr[i]
6) i[arr]
7) *(arr+i)
8) *(arr+i)

i) How [] is working with pointer ?

ii) How * is working with array ?

★ What is the relation between an array and a pointer.

1)

```
void main()
{
  int arr[5];
  int i;
  for (i=0; i<5; i++)
{
   printf("Enter number : ");
   scanf("%d", arr+i);
}
  for (i=0; i<5; i++)
    printf("\n%d", *(arr+i));
}
```

is this Valid ?
Ans :- Yes ✓

2)

```
void main()
{
  int arr[5];
  int i;
  for(i=0; i<5; i++)
{
  printf("Enter Number:");
  scanf("%d", arr);
   arr++;
}
for(i=0; i<5; i++)
{
   printf("\n%d", *arr);
   arr++;
}
}
```

is this Valid ?
Ans :- No ✗

ERROR will be like ↓

L-Value Required

Date
Page

★ Why arr++ is wrong & p++ works ?

- arr ++ ;

arr = arr + 1 ;
arr = 2000 + 1 ;
arr = 2002

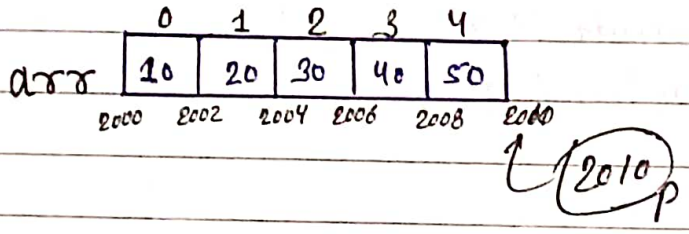
10 = 10 + 1 ;
10 = 11 ;

L-value Required

Constant

- p++ ;

p = p + 1 ;
p = 2000 + 1 ;
p = 2002 ;



★ What is an array for C Language compiler?

For the C compiler, an array is always a constant pointer

Constant means :-

Neither we can increment nor decrement this address.

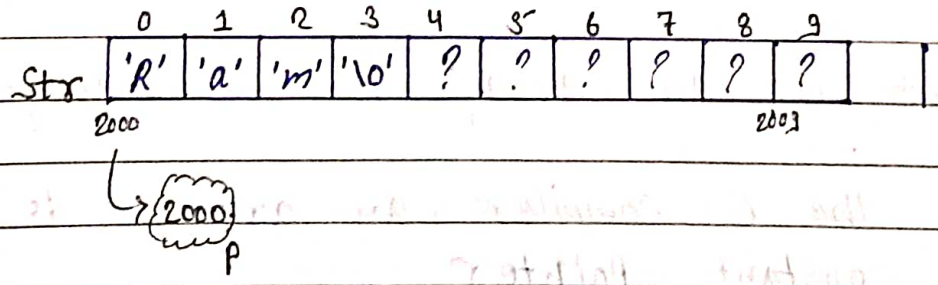
Because array name also produces address

L-value should always be variable

L-value ——> x = y; <—— R-Value

☆ Accessing character array Using Pointer

```
Void main()
{
 char str[10], *p;
 p= str;
 printf("Enter name");
 scanf("%s", p);
 printf("%s", p);
 getch();
}
```



Date
Page

✱ Traversing A character Array using Pointer

without using pointer :-

```
Void main()
{
  Char str[10], *p;
  int i;
  p = str;
  printf("Enter name");
  scanf("%s", p);
  for(i=0; str[i]!='\0'; i++)
      printf("\n%c", str[i]);
  getch();
}
```

First way :-

```
Void main()
{
  Char str[10], *p;
  int i;
  p= str;
  printf("Enter name");
  scanf("%s", p);
  for( i=0; *(p+i)!='\0'; i++)
      printf("\n%c", *(p+i));
  getch();
}
```

Second way :—

```
void main()
{
 char str[10], *p;
 p= str;
 printf("Enter name");
 scanf("%s", p);
 for(; *p != '\0'; p++)
     printf("\n%c", *p);
 getch();
}
```

Using while loop :—

```
void main()
{
 Char str[10], *p;
 p= str;
 printf("Enter name");
 scanf("%s", p);
 while (*p != '\0')
 {
     printf("\n%c", *p);
     p++;
 }
 getch();
}
```

Some New Shortcuts :—

1)
```
int a, b, c;

a = 10;
b = 10;
c = 10;
  or
a = b = c = 10;
```

2)
```
Char Ch = 'A';
    or
Char Ch = 65;
```

3)
```
int a = 10;
    or
a = a+1;
  or
a += 1;
  or
a++;
  or
++a;
```

Date
Page

★ Some New Shortcuts :—

2)

```
if(<var_name> != 0)
{
  - - - -
}
```

or

```
if(<var_name>)
{
  - - -
}
```

Ex :—

```
if( a %2 != 0)
    printf(" odd No.");
```

or

```
if( a %2 )
   printf(" odd No.");
```

Ex :—

```
while (a+b)
{
  - - - -
}
```

This means that the loop should run until the sum of a and b becomes 0

★ Shortcuts to Run Infinite Loop :—

Can you run an infinite for loop?

```
for(;;)
{
}
```

- Can we do the same thing with the while ?

```
while( )  <-- ERROR
{
  ---
}
```

- But now do we have some way ?

```
while( 1 != 0)  <--
{
  ---
}
```

ok this means while(1!=0) and this condition will never become false

☆ Guess the output :—

```
Void main()
{
  int a = 10;
  if( a = 5)
     printf(" Hi ");
  else
     printf(" BYE ");
  printf("\n a = %d ", a);
  getch();
}
```

if(a = 5-)

↓

```
if((a = 5) != 0)
{
    printf(" Hi ");
}
```

The output :—

Hi

a = 5-

Date
Page

★ Some New Short cuts :—

2)
```
if( <var_name> == 0)
{
  -- --
}
```

Or

```
if( !<var_name>)
{
  -- --
}
```

Ex :—
```
if( a==0)
{ ---
}
```
Or
```
if( !a)
{ --
}
```
→ SAME

Logical Not operator

Ex :—
```
if( strcmp(arr, brr) == 0)
{
  ----
}
```

Or

```
if( !strcmp(arr, brr))
{
  ---
}
```
→ Same

Date
Page

★ Previous Code Using Shortcut :-

```
Void main()
{
  Char str[10], *p;
  p = str;
  printf("Enter name");
  scanf("%s", p);
  while (*p != '\0')
{
   printf("\n%c", *p);
   p++;
}
getch();
}
```

→ Same meaning for compiler

OR

```
Void main()
{
  Char str[10], *p;
  p = str;
  printf("Enter name");
  scanf("%s", p);
  while (*p)
  {
      printf("\n%c", *p);
      p++;
  }
  getch();
}
```

for Example :—

while (*p != 'A')	while (*p != '\0')
or	or
while (*p != 65)	while (*p != 0)

for (i=0; *(p+i) != '\0'; i++)
or
for (i=0; *(p+i); i++)

} → Same meaning for compiler.

" Exercise "

Q→ WAP to accept a string from the user and store it in a character array and find out the length of the string.

MAKE SURE YOU ARE ALLOWED TO USE ONLY AN ARRAY AND A POINTER, NOTHING ELSE

```
void main()
{
    char str[10], *p;
    p = str;
    printf("Enter name");
    scanf("%s", p);
    while(*p)
    {
        p++;
        printf("\nLength is = %d", p-str);
        getch();
}
```

Output :-

Enter name Vineet
Length is = 6

A There is a special rule regarding Subtraction of address :-

whenever we subtract two address or two pointer, then we never get the actual answer Rather the compiler divides the actual answer with the sizeof(data type of address or pointer and then returns the answer.

Ex :-

```
int arr[5];
int *p, *q;
p = arr;        // 3000
q = &arr[3];    // 3000
```

printf("%d", q - p);

* other then Subtraction, no arithmetic operation is allowed between nature 2 pointer;

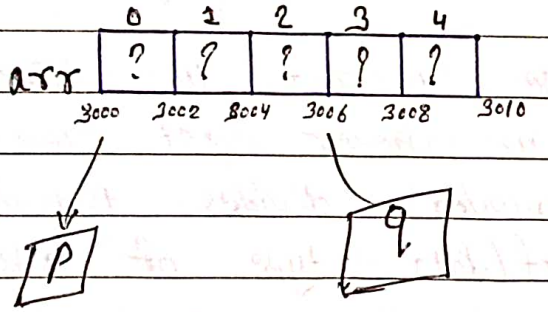
1) p + q → ERROR

2) p * q → ERROR

3) p / q → ERROR

4) p % q → ERROR

5) p - q → Allowed, that too only when pointers are of same data type



" PASS by reference

Date
Page

☆ why ~~are~~ we need pass by reference :-

```
void increment (int, char);
void main()
{
   int a = 10;
   char ch = 'A';
   printf(" Before calling function increment : \n");
   printf(" value of a : %d \n value of ch : %c \n", a, ch);
   increment(a, ch); <-->  increment(10, 'A');
   printf(" After calling function increment : \n");
   printf(" value of a : %d \n value of ch : %c \n", a, ch);
   getch();
}
void increment (int x, char y)
{
   x = x+1;
   y = y+1;
}
```

actual argument

formate argument

call by value

Output :-

```
Before calling function increment :
value of a : 10
value of ch : A
After calling function increment :
value of a : 10
value of ch : A
```

Date
Page

Point to Remember :-

→ Limitations with call by value :-

- Changes done to the formal argument are never reflected back to the actul argument if we are using pass by value
- If we want to changes done on formal argument reflect back in the actual argument then we should go with call by refrence.

Call by reference :-

```
void increment (int *, char *);
void main ()
{
    int a = 10;
    char ch = 'A';
    printf("Before calling function increment: \n");
    printf("value of a: %d \n value of ch: %c\n", a, ch);
    increment (&a, &ch); ↔ increment(1000, 2000);
    printf("After calling function increment: \n");
    printf("Value of a: %d \n Value of ch: %c \n", a, ch);
    getch();
}
void increment (int *x, char *y)
{   *x = *x + 1;
    *y = *y + 1;
```

actual argument

Call by reference or Pass by reference

Output :-

Before calling function increment :
value of a : 10
value of ch : A
After calling function increment :
value of a : 11
value of ch : B

A Point To Remember :-

→ Application of call by reference :-

we go with the call by reference when changes done to formal arguments are need to be reflected back to actual arguments :-

In C Language there are 2 types of argument passing :-

1) Pass by value

2) Pass by Reference

A why we call scanf() using "&" operator.

i) int a;
scanf("%d", &a);

- call by reference
(It will change the value of

V/S

ii) int a;
scanf("%d", a);

- call by value
(It will never change the value of

This is because scanf() works on call by reference i.e., we have to pass the address of the variable to scanf() only then it can change its contents.

Exercise :—

1) WAP to declare 2 variables called a and b of type int in main(). Accept input from the user in them and pass them to a function called swap().

within swap() exchange the values of the variables but print back the new value in main().

Solution 1 :—

```
void swap( int* , int*);
void main()
{
   int a, b;
   printf("Enter 2 int");
   scanf("%d %d", &a, &b);
   printf("Before Swapping a = %d, b = %d \n", a, b);
   swap(&a, &b);
   printf("After swapping a = %d, b = %d \n", a, b);
   getch();
}
void swap( int *p, int *q)
{
   int temp;
   temp = *p;          //  *p = *p + *q;
   *p = *q;            //  *q = *p - *q;
```

```
    *q = temp;      // *p = *p - *q;
}
```

2) WAP to accept the radius of circle from the user in the function main(), Now pass it to a function called calculate() within the function calculate(), fide out the area and circumference of the circle, but display the result in the function main().

Hint :— function declaration

void calculate (int, float*, float*)

3) WAP to create a function called checkprime() which should accept an integer as an argument and return 1 if it is prime and 0 if it is not prime.

Finally the function main() should display the message whether the number is prime or not.

4) WAP to create a function called minmax() which should continuously accept Integer from the user until the user inputs 0. As soon as 0 is inputted, the function should display the minimum and the maximum numbers amongest all the numbers given before 0.

5) WAP to create a function called fibo() ~~which~~ which should accept an integer as an argument and should print Fibonacci series terms up to the given integers. Fibonacci series is an infinite series :

0, 1, 1, 2, 3, 5, 8, 13, 21 .....

★ Passing Arrays as an argument to functions :—

• Till now we have discussed passing :

Passing Variables to Functions

Passing address of variables to functions

Now we will discuss passing array as argument to functions

⇒
```
Void display (int *);
Void main()
{
    int arr[5];
    int i;
```

```
        for(i=0; i<5 ; i++)
        {
            printf(" Enter element : ");
            scanf("%d ", &arr[i]);
        }
        display (arr);
        getch ();
}
void display (int *p)
{
    int i;
    for (i=0 ; i<5 ; i++)
        printf("\n%d ", *(p+i));
}
```

Output :-

```
Enter element : 10
Enter element : 20
Enter element : 30
Enter element : 40
Enter element : 50

10
20
30
40
50
```

Date
Page

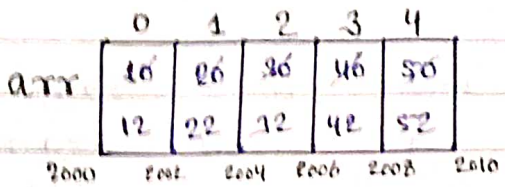
★ Guess the output :-

```
void display (int *);
void main()
{
   int arr[5];
   int i;
   for (i=0; i<5; i++)
   {
      printf(" Enter element : ");
      scanf(" %d ", &arr[i]);
   }
   display (arr);
   for (i=0; i<5 ; i++)
    printf("\n%d", arr[i]);
   getch();
}
void display (int *p)
{
  int i;
  for (i=0 ; i<5 ; i++)
  {
     printf("\n %d ", *(p+i));
    *(p+i) = *(p+i) +2 ;
  }
}
```

Output :-

Enter element : 10
Enter element : 20
Enter element : 30
Enter element : 40
Enter element : 50

10
20
30
40
50
12
22
32
42
52



Call / Pass By Reference

A few Important points about arrays :

1) Name of an array always represents its base address

2) whenever we pass an array as an argument to a function, always its address will be passed and to recive that address we will require a pointer as a formal argument

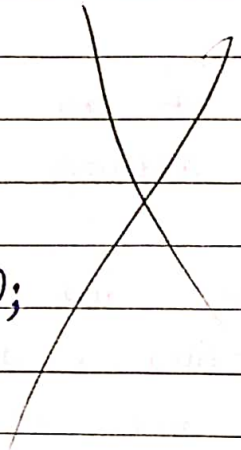
3) Since whenever we pass an array, the Compiler passes its base address so we say that it always PASS / CALL By Referenc

4) in other words we can say arrays can never be passed using Pass By Value.

5) If we will make any change in the array using a formal argument inside the function to which we have passed this array, then the changes that have been done there will always be reflected in the actual array.

* second style of array passing :-

```
void display(int[]);
void main()
{
  int arr[5];
  int i;
  for(i=0; i<5; i++)
  {
    printf("Enter element : ");
    scanf("%d", &arr[i]);
  }
  display(arr);
  getch();
}
void display(int brr[5])
{
  int i;
  for(i=0; i<5; i++)
  printf("\n%d", brr[i]);
}
```



6) If we declare an array as a formal argument, then the compiler automatically converts it into POINTER. Thus arrays can never be used as Formal Argument.

☆ EXERCISES :-

1) WAP to create a function called mystrlen which should work exactly same as the library function strlen().

Hint :- int mystrlen(char*);

Solution :-

```
int mystrlen(char*);
void main()
{
    char str[10];
    int x;
    printf(" Enter a string : ");
    gets(str);
    x = mystrlen(str);
    printf(" \n Length = %d ", x);
    getch();
}
int mystrlen(char *p)
{
   int i;
   for(i=0; *(p+i) != '\0'; i++);
   return i;
}
```

Output :—

```
Enter a string : Hello
Length = 5
```

2) WAP to create a function called mystrcpy(), which should work exactly same as library function strcpy().

Hint :— void mystrcpy(char*, char*);

Solution :—

```
void mystrcpy(char*, char*);
void main()
{
    char arr[10], brr[10];
    printf("Enter a string : ");
    gets(arr);
    mystrlen(brr, arr);
    printf(" arr : %s \n", arr);
    printf(" brr : %s \n", brr);
    getch();
}
void mystrcpy(char*p, char*q)
{
    int i;
    for(i=0; q[i] != '\0'; i++)
    {
        p[i] = q[i];
    }
    p[i] = '\0';
}
```

Output :-

```
Enter a string : Hello
arr : Hello
brr : Hello
```

Solution 2 :-

```
void mystrcpy( char*, cha* );
void main()
{
   Char arr[10], brr[10];
   printf(" Enter a string : ");
   gets(arr);
   mystrcpy(brr, arr);
   printf(" arr : %s \n", arr);
   printf(" brr : %s \n", brr);
   getch();
}
void mystrcpy (char*d, char*s)
{
   for( ; *s != '\0' ; s++,d++)
        *d = *s;
   *d = '\0';
}
```

Output :-

```
Enter a string : Hello
arr : Hello
brr : Hello
```

Date
Page

Dynamic Memory Allocation :-

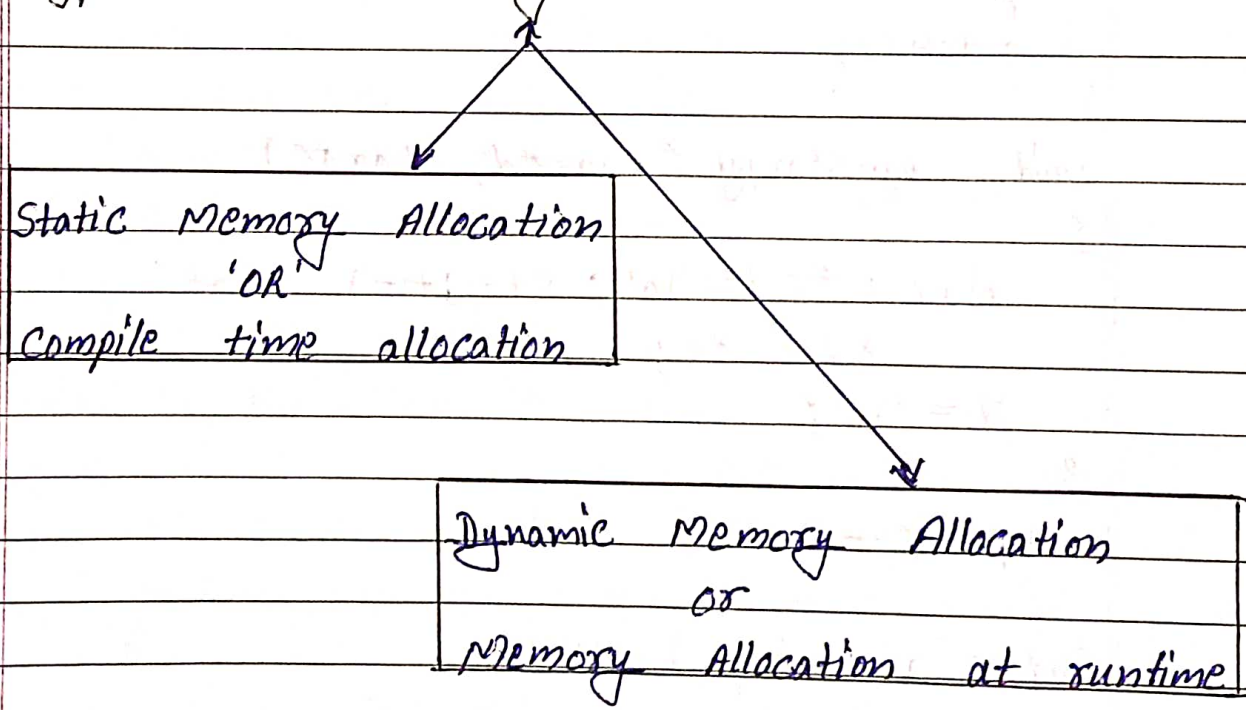
In computer science the meaning of the "Dynamic" is activities that happen at runtime

Then ~~Dat~~ Dynamic Memory Allocation will be termed as Memory allocation at run time.

Dynamic :- Things which happen ~~in~~ at runtime.

Dynamic Memory allocation :- Memory allocation at runtime.

Types of Memory Allocation :-



1) int a; ⟶ 2 Byte
int arr[10]; ⟶ 20 Byte

All calculation for variables / arrays declared in this way are done at compile time thus, this type of calculation is known as static memory allocation or compile-time allocation.

2) Dynamic Memory Allocation :-

- Concerning array, the programmer can't always decide the size of the array that will be required.

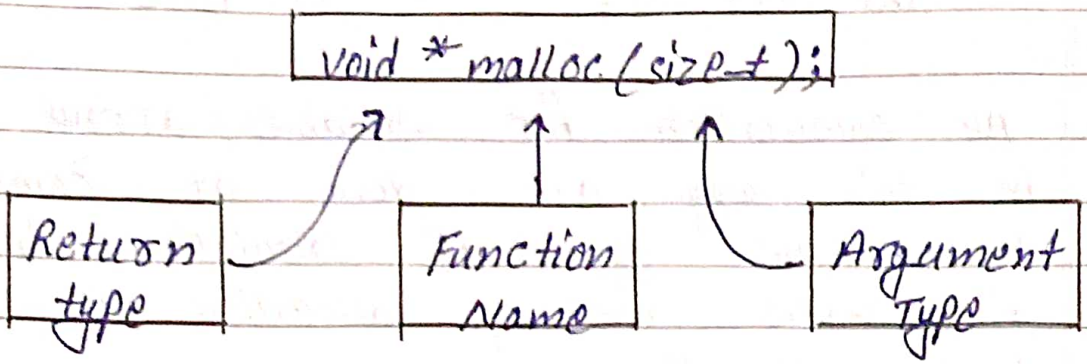
- If we want to create a user-defined size of the array then we should go for Dynamic Memory Allocation.

How is Dynamic Memory Allocation performed in C :-

- In C language to allocate memory at runtime, we have to call a predefined function called malloc()

Header File : alloc.h

Prototype of function malloc();



☆ What is size_t ?

- To understand size_t we first have to understand how can we alias a data type in C language by the user defined ~~inde~~ identifier.

For this we have to study a concept known as typedef in C language.

☆ Using the keyword "typedef" :-

- Syntax of typedef :

typedef <old_data_type><new_name>;

Ex:-

typedef int number;

we are telling the compiler that the data type int should also be named as the number.

Ex:-

```
typedef int number;
void main()
{
    number a;
    number b;
    int c;
    :
    :
}
```

size_t :-

There is a typedef mentioned by the C language compiler :

```
typedef unsigned int size_t;
```

C language itself has given 2 names to the data type unsigned int

1) unsigned int
2) size_t

This line is mentioned in the following header files :

1) alloc.h
2) stdlib.h
3) string.h

★ what is a void Pointer ?

```
int a = 10;
char b = 'x';
float c = 1.5f;
```

■ How many pointers do we require to access the following variable via a pointer ?

Ans :- we require 3 pointers because we have 3 variables of 3 different data type.

■ By using a single void pointer we can access all the above variables

```
void *p;
p = &a;  ✓
p = &b;  ✓
p = &c;  ✓
```

void pointer is a special TYPE of pointer which can point to any variable irrespective of its data type

Date
Page

★ How the pointer access the value of a variable :-

1)
```
int a = 10;
int *p;
p = &a;
printf(" %d ", *p); ⟶ 10
```

2)
```
void *p;
p = &a;
```

we cannot dereference a void pointer in a normal way

```
int a = 10;
void *p;
p = &a;                    ✗ ↷ ERROR
printf(" *p : %d \n ", *p);
```

Solution :-

```
int a = 10;
void *p;
p = &a;
printf(" %d \n ", *(int*)p); ⟶ 10
```

■ If we want to again dereference it then again we have to type cast because type casting effect is Temporary.

printf("%d\n", *(int *)p)); → 10

Suppose q is a pointer, then how many bytes will q++ jump

q++; → Depends on it's data type

int → 2 Byte
Char → 1 Byte
float → 4 Byte
void → Error X

q++ → q = q + 1 * sizeof(<data_type_of_q>);

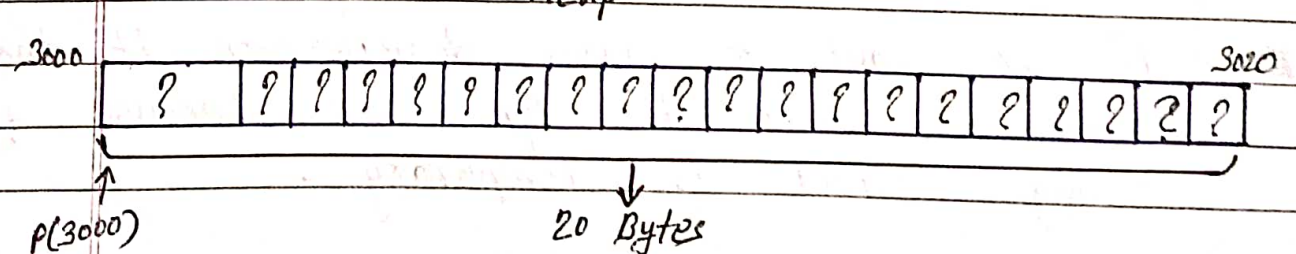
★ Functionality of malloc() :-

void * malloc(size_t);

size_t ← unsigned int ← Number of Bytes

Ex:-

void *p = malloc(20); ← Give me an array of 20 Bytes



Date
Page

★ Dynamic Memory allocation :-

function :- malloc()

Header file :- alloc.h

Prototype :- void * malloc(size_t);

unsigned int (size) → Number of bytes

void * → Base Address of the dynamic block

★ Predict the output of the code :-

```
void main()
{
    void *p;
    p = malloc(10*sizeof(int));
    *p = 10; ← ERROR
}
```

Because we cannot dereference a void pointer without type casting.

Date
Page

* Predict the output of the code :-

```
void main()
{
  void *p ;
  p = malloc (10 * sizeof(int));

  *p = 10; X ERROR
   p++; X ERROR
}
```

Cause we cannot dereference a void pointer without type casting.

Because we cannot increment / decrement a void pointer without type casting

\# Solution of the previous code :-

```
void main()
{
   int *p;
   p = (int*) malloc (10*sizeof(int));
   *p = 10;
   p++;
```

Replace with 'n' where 'n' is a variable

Now, they are perfectly valid, because p is not a void pointer, rather it is an int pointer

" Exercise "

1) WAP to create a dynamic array of n integers, where n should be taken from the user, Then ask the user to input values in the array and finally display all the numbers inputted along with their sum and average.

Solution:-

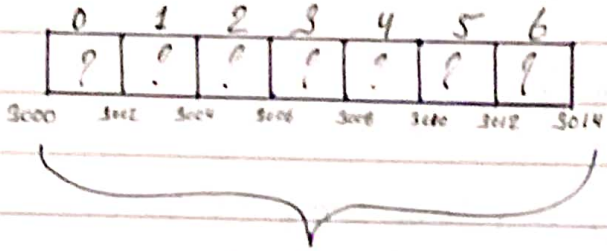
```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <stdlib.h>

void main()
{
    int i, n, sum = 0; *p;
    printf("Enter how many numbers of elements
            you want to store ? : ");
    scanf("%d", &n);
    p = (int*)malloc(n*sizeof(int));
    if(p = NULL)
    {
        printf("Sorry ! Insufficient memory \n");
        exit(1);
    }
    for(i=0; i<n; i++)
    {
        printf("Enter a number : ");
        scanf("%d", p+i);
    }
```

```
for(i=0;i<n;i++)
{
    sum = sum + *(p+i);
    printf("\n %d", *(p+i));
}
free(p);
printf("\n Sum : %d", sum);
printf("\nAverage : %f", sum/(float)n);
getch();
}
```

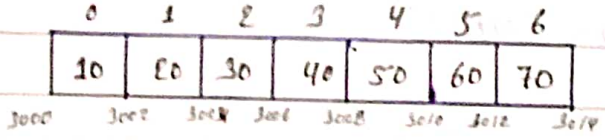
Memory Diagram of the previous code

Before the input taken from the user



14 Bytes

After the input taken from the user



Date
Page

Sample input / output (First)

```
Enter how many number of elements you want to store?:7
Enter a number : 10
Enter a number : 20
Enter a number : 30
Enter a number : 40
Enter a number : 50
Enter a number : 60
Enter a number : 70

10
20
30
40
50
60
70
Sum : 280
Average : 40.000000
```

(second attempt):-

```
Enter how many numbers of elements you want to store:-
Sorry ! Insufficient memory

Process returned 1 (0x1) execution time : 4.022s
Press any keys to continue.
```

(a Third attempt :-)

Enter how many number of elements you want to store: 0

Sum : 0
Average :- -1. #IND00

"Structure"

- To understand the concept of "Structure".

we first try to find a solution to a given problem through C programming.

- we have to take a student's name, age, and percentage in our program and display it back on the screen using the knowledge we have gained till now.

Solution to the above given problem :-

```
int main()
{
    int roll;
    char grade;
    float per;
    printf("Enter roll : ");
```

```
    scanf("%d", &roll);
    fflush(stdin);
    printf("Enter grade : ");
    scanf("%c", &grade);
    printf("Enter Percentage : ");
    scanf("%f", &per);
    printf("Roll : %d \n", roll);
    printf("Grade : %c \n", grade);
    printf("Percentage : %f \n", per);
    return 0;
}
```

⇒ Now, Suppose we take the same details for 5 students what will be the solution now.

Solution :—

```
include <stdio.h>
int main()
{
    int roll[5], i;
    char grade[5];
    float per[5];
    for(i=0; i<5; i++)
    {
        printf("Enter roll : ");
        scanf("%d", &roll[i]);
        fflush(stdin);
        printf("Enter grade : ");
        scanf("%c", &grade[i]);
        printf("Enter percentage : ");
        scanf("%f", &per[i]);
    }
```

```
for(i=0; i<5; i++)
{
    printf("Roll : %d\n", roll[i]);
    printf("grade : %c\n", grade[i]);
    printf("percentage : %f\n", per[i]);
}
return 0;
}
```

Q:- Can we take the details in a single array :-

Ans:- No, Because the array is a collection of similar kinds of data types stored at a continuous memory location.

⇒ But we have data of 3 different data types.

Thankfully C language provides us the concept of structure using which we can take a single array to manage the details of 5 different students.

Defination :- A structure is also a collection of dissimilar kinds of data elements stored at continuous memory locations

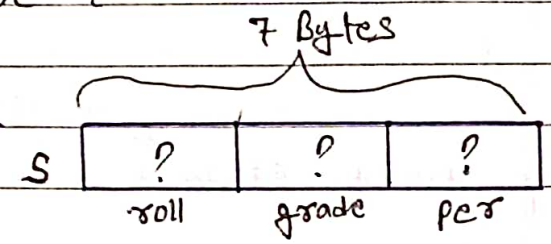
Syntax of Declaring a Structure :-

```
Struct <Struct_name>
{
    <data_type> <variable_name>;
    <data_type> <variable_name>;
    <data_type> <variable_name>;
    :
    :
};
```

Using a Structure :-

1)

```
Struct Student
{
    int roll;
    char grade;
    float per;
};
int main()
{
  Struct Student s;
   // code
  return 0;
}
```



2)

```
struct Student
{
    int roll;
    char grade;
    float per;
};

int main()
{
  struct Student S;
  roll = 10; ← X  Error
  return 0;
}
```

3)

Tag or Name

```
struct Student
{
    int roll;
    char grade;
    float per;
};
int main()
{
  struct Student S;
  S.roll = 10;
  S.grade = 'A';
  S.per = 76.5f;
  printf("Roll : %d , Grade : %c , Percentage : %f\n",
          S.roll, S.grade, S.per);
  return 0;
}
```

Declaration or Template

members

Structure variable

Data type Name

Output :-

Roll : 10 , Grade : A ; Percentage : 76.500000

Three Important Points to Remember :-

1) we can never initialize members of a structure inside the declaration block of the structure

```
Struct student
{
 int roll = 10;
 char grade = 'A';
 float per = 76.5f;
};
```

Error.

Reason :- Roll , grade , and per will be stored in memory when the line "struct student" is executed .

2) After declaring the template of the structure we must terminate it with a semicolon . Otherwise , the compiler will generate a SYNTAX ERROR .

```
Struct student
{
   int roll;
  char grade;
   float per;
```

};

Because we are declaring structure that's why they are ~~ness~~ necessary and they are necessary and the part of syntax and always remembers declaration always terminated by ";" in C language

3) The Keyword Struct must compulsorily be used whenever we write the tag/name of the structure in our program. otherwise, the compiler will give the error.

~~Stdut~~ Student s; ✗ ERROR
Struct student s; ✓ correct

★ Using the Keyword "typedef" with structure :-

```
Struct student
{
int roll ;
char grade;
float per ;
};
typedef struct student student ;
int main ()
{
```

```
    Student S;
    :
    ;
}
```

/OR/

```
typedef struct student
{
    int roll;
    char grade;
    float per;
} student;

int main()
{
    Student S;
    :
    ;
}
```

Now, No error will appear because we have assigned an alternative name to struct student i.e., Student.

Using Initializer List with Structure

1)

```
Struct Student
{
  int roll;
  char grade;
  float per;
};
int main()
{
  struct student s = {10, 'A', 76.5f};
  printf("Roll : %d, Grade : %c, Percentage : %f \n",
        s.roll, s.grade, s.per);
  return 0;
}
```

Output :-

Roll : 10, Grade : A, Percentage : 76.500000

2)

```
struct Student
{
   int roll;
   char grade;
   float per;
};
int main()
{
   struct Student s;
```

```
printf("Enter roll, grade and percentage : ");
scanf("%d %c %f", &s.roll, &s.grade, &s.per);
printf("Roll : %d, Grade : %c, Percentage : %f \n",
        s.roll, s.grade , s.per );
```

★ Copying One Structure variable To Another :-

```
Struct student
{
    int roll;
    char grade;
    float per;
};
Int main()
{
    struct student s, p;
    printf("Enter roll, grade and per : ");
    scanf("%d %c %f", &s.roll, &s.grade, &s.per);
    p.roll = s.roll;
    p.grade = s.grade;
    p.per = s.per;
    return 0;
}
```

`p = s;` //single line shortcut

In c language we can use assignment operator on structure also provided these structure variables are of same type

★ Creating Array of structure

```
Struct Student
{
   int roll;
   char grade;
   float per;
};
int main()
{
   Struct Student s[3];
   for( i=0; i<3; i++)
    {
      printf("Enter roll, grade and per: ");
      Scanf("%d.%c %f", &s[i].roll, &s[i].grade, &s[i].per);
    }
   for( i=0 ; i<3 ; i++)
   Printf("%d %c %f\n", s[i].roll, s[i].grade, s[i]. per );

    return 0;
}
```

★ Structures and Pointers :—

```
struct student
{
   int roll;
   char grade;
   float per;
};
```

```
int main ()
{
    Struct student S;
    Struct student *p;
    p = &S;
    p -> roll = 10;
    p -> grade = 'A';
    p -> per = 76.8f;
    printf("%d %c %f", p->roll, p->grade, p->per);
    return 0;
}
```

Arrow operator

1. using structure variable S.roll
   or
2. Using structur pointer p->roll

★ Accepting Input in structure using pointer :-

```
Struct student
{
    int roll;
    char grade;
    float per;
};
int main()
{
    Struct student S;
    struct student *p;
    p = &S;
    printf("Enter roll, grade and per : ");
```

```
    scanf("%d %c %f", &p->roll, &p->grade, &p->per);
    printf("%d %c %f\n", p->roll, p->grade, p->per);
    return 0;
}
```

Array of Structure and Pointer :-

```
struct student
{
  int roll;
  char grade;
  float per;
};
int main()
{
   struct student s[3];
   struct student *p;
   p = s;
   int i;
   for (i=0; i<3; i++)
   {
    printf("Enter roll, grade and per : ");
    scanf("%d %c %f", &(p+i)->roll, &(p+i)->grade,
          &(p+i)->per);
   }
   for(i=0; i<3; i++)
   printf("%d %c %f\n", (p+i)->roll, (p+i)->grade,
          (p+i)->per);
   return 0;
}
```

p+i ⇒ p + i * 9

2000 + 0 * 9 = 2000

2000 + 1 * 9 = 2009

2000 + 2 * 9 = 2018

☆ Structure and functions :-

Pass By Value :-

```
struct student
{
   int roll ;
  char grade ;
  float per ;
};
void display (struct student);
int main()
{
   struct student s;
   printf(" Enter roll, grade and per : ");
   scanf(" %d %c %f ", &s.roll, &s.grade ,&s.per );
   display (s);
   return 0;
}
Void display ( struct student p )
{
     printf(" %d %c %f \n", p.roll, p.grade , p.per );
}
```

Date
Page

Pass By Reference :-

```
Struct student
{
    int roll;
    char grade;
    float per;
};
Void accept (struct student *);
void display (struct student);
int main()
{
    Struct student s;
    accept(&s);
    display(s);
    return 0;
}
void accept(struct student *p)
{
 printf("Enter roll, grade and per:");
 scanf("%d %c %f", &p->roll, &p->grade, &p->per);
}
void display (struct student p)
{
 printf("%d %c %f\n", p.roll, p.grade, p.per);
}
```