Syntax :- function declaration

returntype functionname(arguments);

4 ways to design a function

1. No arguments No return type

Void fun_name(Void);

This function will not take any arguments & will not return any value to calling function.

For eg, Void abc(Void);

2. Arguments but No return type

Void fun_name(arguments);

This function will take one or more arguments but will not return anything to calling function

For eg, Void print_Tel(int batch, int rollno);

3. Return type but no arguments

return_type fun_name(Void);

4. Arguments with return type

return_type fun_name(arguments);

This function will take arguments and return value to calling function. For eg, int sum(int a, int b);

Simple example illustrating function declaration, function call and function definition.

Primenumber.c

Design a function to print binary format of given number.

Actual arguments → Arguments passed at the time of function call.
formal arguments → Arguments passed at the time of definition.

```
#include<stdio.h>
Void print_binary(int);
Void main()
{
  int num;
  Pf("Enter the number\n");
  Sf("%d",&num);
  Print_binary(num); // function call
  Printf("Hello\n");
}
Void print_binary(int num)
{
  int pos;
  for(pos=31;pos>=0;pos--)
  Pf("%d",num>>pos&1);
  Pf("\n");
}
```

Program control

// once the function is completed, the program control goes to the main()

Design a function to count how many bits are set in a given no.

```
#include<stdio.h>
Void print_binary(int);
int count_set_bits(int);
Void main()
{
  int num, c;
  Pf("Enter number\n");
  Sf("%d",&num);
  print_binary(num);
  c=count_set_bits(num);
  Pf("c=%d\n",c);
}
```

```
int count_set_bits(num n)
{
  int c,i;
  for(i=0,c=0;i<31,i++)
    if(n&i<<1)
      c++;
      return c;
}
void print_binary(int num)
{
   int pos;
  for(pos=31;pos>=0;pos--)
  pf("%d",num>>pos&1);
  pf("\n");
}
```

Design a function to check whether the given number is prime or not. If prime return 1 else '0'

```
#include<stdio.h>
 int prime_check(int n)
 {int i;
  for(i=2;i<n;i++)
       if(n%i==0)
        break;
 if (n==i)
 return 1;
 else
  return 0;
}
void main()
{int r,num;
 pf("Enter the number\n");
 sf("%d",&num);
 r= prime_check(num);
 if (r==1)
  pf("Prime\n");
else
pf("Not prime\n");  }
```

Session – 49

14-8-20

Functions

Though the variable names of actual & formal arguments are same, the addresses of these arguments are differed

This is called "call by value"

```
eg. void abc(int);
    void main()
    {
     int i=10;
     fn("In main %i = %p\n",%i);
      abc(i);
    }
    void abc(int i)
    {
     printf("In abc %i = %p\n",%i);
    }
```

Value o/p:-
In main %i = 1000
In abc %i = 1008

Swapping of two numbers (Call by value)

```
void swap(int i, int j)
{
 int t;
  t=i;
  i=j;
  j=t;
}
void main()
{
 int i=10, j=20;
 fn("Before swap i=%d j=%d\n",i,j);
  swap(i,j);
 fn("After swap i=%d j=%d\n",i,j);
}
```

Call by value

→ When the values are passed during fun. call, the value are not swapped.

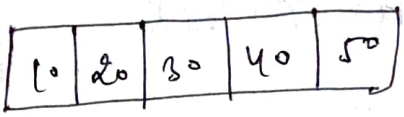
Call by reference

```
void swap(int *p, int *q)
{
 int t;
  t=*p;
  *p=*q
  *q=t;
}
```

On passing the address the variables are swapped.

```
Void main()
{
  int i=10; j=20;
  Pf("Before swap i= %d j= %d \n",i,j);
    swap(&i,&j);
    Pf("After swap i=%d j= %d\n",i,j);
}
```

Design a program to print an integer array




```
#include<stdio.h>
void print_int_array(const int *, int);
void main()
{
  int a[5]={10,20,30,40,50},ele;
  ele = sizeof(a)/sizeof(a[0]);
  print_int_array(a,ele);
  Pf("Done \n");
}
Void print_int_array(const int *p, int n)
{ int i;
for(i=0;i<n;i++)
  Pf("%d",p[i]);
Pf("\n");
}
```

→ array base address is passed

const is used to protect the array from this fun: so that fun will not have permission to modify the array

Calling completion the 'p' points to a's base address only

Pointer in the form of array is used to print

Pf("%d", *p+1); ← here p is inc [if a's base address is 1000, after completion of this, p will point to 1002]

This array base address is caught with a read only pointer

*p[i] = *(p+i) → X error

Design a function for bubble sort

```
#include<stdio.h>
Void bubble_sort(int *, int);
Void main()
{
  int a[5]={50,75,35,20,10},ele;
  ele = sizeof(a)/sizeof(a[0]);
  Pf("Before sort \n");
  Print_int_array(a, ele);
  bubble_sort(a, ele);
  Pf("After sort \n");
}
  Void bubble_sort(int da, int ele)
  {
    int i, j, t;
      for(i=0; i<ele-1; i++)
        {
          for(j=0; j<ele-1-i; j++)
            {
              if(a[j] > a[j+1]
              {
                t= a[j];
                a[j]= a[j+1];
                a[j+1]= t;
              }
        }
  }
```

Design a function to print the string char by char.

```
Void print_string(const char *);
Void main()
{
  char s[20];
  Pf("Enter string \n");
  Sf("%s", s);
```

```
    print_string(s);
}
void print_string(const char *p)
{
    while(*p)
    printf("%c", *p++);
    printf("\n");

    int i;
    for(i=0; p[i]; i++)
    printf("%c", p[i]);
    printf("\n");
}
```

15 Aug
15-8-20

Session - 50

Functions:

Design a function to print the string length.

```
#include<stdio.h>
int my_strlen(const char *);
void main()
{
    int len;
    char s[20];
    printf("Enter the string\n");
    scanf("%s", s);
    printf("len = my_strlen(s);
    printf("Length = %d\n", len);
}
int my_strlen(const char *p)
{
    int i;
    for(i=0; p[i]; i++);
    return i;
}
```

Design a function for copying the source string into the destination.

```
#include<stdio.h>
void my_strcpy(const char *, char *);
void main()
{
  char s[20], d[20];
  Pf("Enter the string\n");
  Sf("%s",s);
  my_strcpy(s,d);
  Pf("s=%s d=%s\n",s,d);
}
void my_strcpy(const char *s, char *d)
{
  while(*s)
  {
    *d= *s;
    s++;
    d++;
  }
  *d= *s; // for '\0'
}
```

these two address are caught by the destination is not made constant because we need to write in place of destination.

source address should be large & destination should be smaller when compiled to if.

eg: my_strcpy(s+1,d);

when this is passed o/p is

s= hello
d= ello

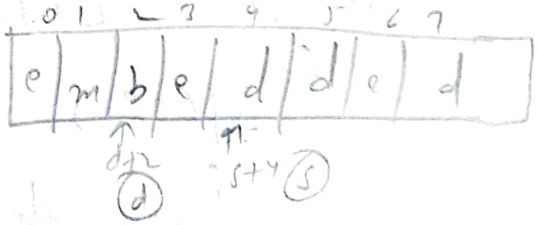
my_strcpy(s+4, s+2)

s= embedded
d= emdded

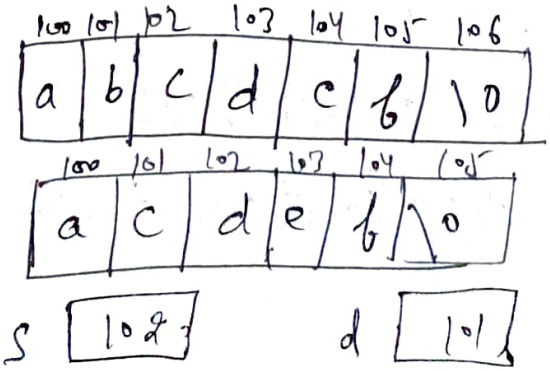
to delete 'e' alone

my_strcpy(s+2, s+1)

then



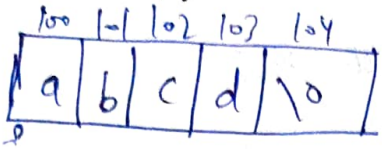
s+2, s+1



The content at 102, content at 101, likewise all the elements arrange themselves due to the shift one char is deleted 'b'

Now, if d > s,

my_strcpy(s, s+1);



s [100] d [101]

the content at 100 is copied at 101.

the while loop never ends and at one points, it may try to access some unauthorized memory which may lead to segmentation fault.

To delete one specific given char in the given string.

```
void my_strcpy(const char*, char *);
void main()
{
  char s[20], ch;
  int i;
  Pf("Enter the string\n");
  Sf("%s", s);
  Pf("Enter the char\n");
  Sf("%c", &ch);
  Pf("Before s=%s\n", s);
  for(i=0; s[i]; i++)
    {
      if(s[i]==ch)
        {my_strcpy(s+i+1, s+i);   // source, destination
         i--;
        }
    }
  Pf("After s=%s\n", s);
}
void my_strcpy(const char *, char *)
  {
```

```
while (*s)
{
    *d = *s;
    s++;
    d++;
}
*d = *s;
}
```

with for loop

```
int i;
for(i=0;s[i];i++)
    d[i] = s[i];
d[i] = s[i];//this is for '\0'
```

here we are printing incrementing the value of the char array

Session – 5

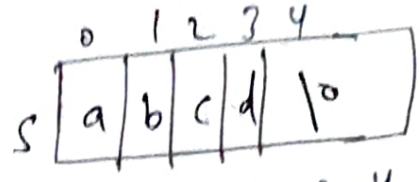
17-Aug 20

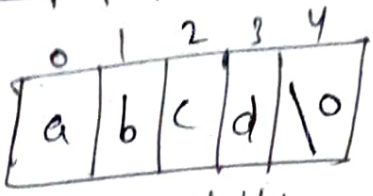
Functions

Design a function to compare 2 strings. If both strings are equal return 0, If 1st string is greater than 2nd return 1, If 2nd string is greater than the 1st string, return -1.

```
#include<stdio.h>
int my_strcmp(const char *, const char *);
void main()
{
    char s[20], s1[20];
    int i;
    Pf("Enter the strings\n");
    Sf("%s %s",s,s1);
    i = my_strcmp(s,s1);
    if(i==0)
    Pf("Both are equal\n");
    else
    Pf("Not equal\n");
}
int my_strcmp(const char *p, const char *q)
{
    int i;
    for(i=0 ; p[i]; i++)
    {
```

```
    if(p[i] != q[i])
        break;
}
if(p[i] == q[i])
    return 0;
else if(p[i] > q[i])
    return 1;
else
    return -1;
}
```



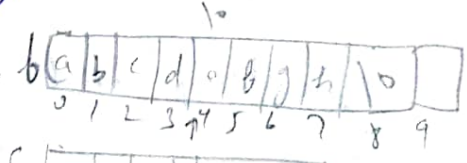


('\0' should match, '\0' should be at the same position in both the pointer)

Design a function to concatenate two strings

Note:- 2nd string should be added at the end of 1st string

```
#include<stdio.h>
void my_strcat(char *, const char *);
void main()
{
    char b[40], s[20];
    printf("Enter 2 strings\n");
    scanf("%s %s", b, s);
    my_strcat(b, s); // fun call
    printf("After b=%s s=%s\n", b, s);
}
void my_strcat(char *b, const char *s)
{
    int i, j;
    for(i=0; b[i]; i++); // i is the length of 1st string
    for(j=0; s[j]; i++, j++)
        b[i] = s[j];
    b[i] = s[j];
}
```



Same task using while loop

strlen()

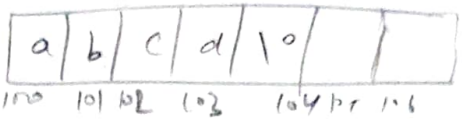
```
while(*b)
  b++;
```

← this will transverse until '\0' in string 'b'

After this statement, the address will be 104

strcpy()

```
while (*s)
{
  *b = *s;
  b++;
  s++;
}
*b = *s;
```



During the 1st while loop, the address is pointed to 104

Instead

```
void my_strcat (char *b, const char *s)
{ int i = my_strlen(b);
  my_strcpy(s, b+i);
}
else
  my_strcpy(s, b+my_strlen(b));
```

// Initially, b will hold the base address (starting address) b+i will give the address where there is '\0'.

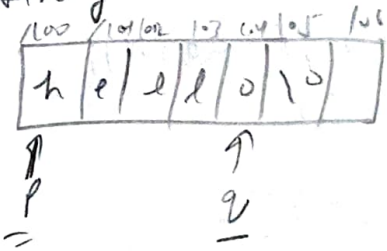
Design a function to reverse the given string (strrev)

```
void my_strrev(char *);
Void main()
{
  char s[20];
  Pf("Enter the string\n");
  Sf("%s", s);
  Pf("Before rev %s\n", s);
  my_strrev(s);
  Pf("After rev %s\n", s);
}
Void my_strrev(char *p)
{
  char ch;
  int i, j;
```

```
for(i=0; p[i]; i++);
for(i=i-1, j=0; i>j; i--, j++)
{
    ch = p[i];
    p[i] = p[j];
    p[j] = ch;
}
```

Same logic using while loop

```
void my_strrev(char *p)
{
  char ch;
  char *q;
  q = p; // Assigning the base address of p to q.
  while(*q)  having till the end of the statement until
  {          q reaches '\0'
   q++;
  }
q--; // Dec the q's position, so that *q != '\0'
 while (p<q)  p holds the starting address of the
  {           string whereas q holds the end of
    ch = *p   the string
    *p = *q
    *q = ch;
   p++;
    q--;
  }
}
```



18-8-20

Session 5 & 6

Design a function to search if a given char is present in a given string. If found, return the location where it is found, else return zero

my_strchr

```
#include<stdio.h>
char *my_strchr(char*, char);
Void main()
{
 char s[20], ch, *p;
 Pf("Enter the string\n");
 Sf("%s",s);
 Pf("Enter the char\n");
 Sf("%c",&ch);
 Pf("Base address =%p\n",s);
  p= my_strchr(s, ch);
  if (p==0)
  Pf("char not present\n");
  else
  Pf("char is present at %p %c %s\n", p, *p, p);
}
char *my_strchr(char *p, char ch)
{
      while (*p)
      {
        if(*p==ch)
          return p;
          p++;
      }return 0;
}
```

will print the string from the char found.

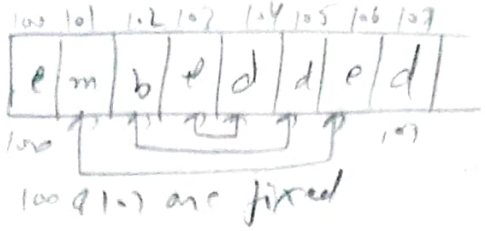
← This program will print only 1st occurance

Using my_strcpy() & my_strchr(), delete a given char in a given string

Inside main

```
while (p=my_strchr(s,ch)
      my_strcpy(p+1, p);
```

Design a function to reverse the data in between two locations.



Reversing a particular portion of a string

```
void my_strrev_1(char *, char *)
void main()
{ char s[50], ch, *p;
  int i, j;
  Pf("Enter the string \n");
  Sf("%s", s);
  Pf("Before s= %s \n", s);
  Pf("Enter the indexes to be made constant \n");
  Sf("%d %d", &i, &j);
  my_strrev_1(s+i, s+j);
  Pf("After s= %s \n", s);
}
void my_strrev_1(char *p, char *q)
{ char ch;
  while (p < q)
  {
    ch = *p;
    *p = *q
    *q = ch;
    p++;
  }  q--;
}
```

To reverse the words in a given string.

```
#include<stdio.h>
void my_strrev_1(char *,char *);
void main()
{
  char s[50], *p, *q;
  Pf("Enter the string \n");
  Sf("%[^\n]",s);
  Pf("Before rev = %s \n",s);
  q=p=s;
     while(p=my_strchr(p,' '))
     {
      my_strrev_1(q,p-1);
       q=p=p+1;
     }
     my_strrev(q);
  Printf("After rev %s \n",s);
}
```

100 104 109 116 122

A B C D ⊥ E F G H I I J K L M N I O P Q R S T U V W X Y Z \0

p	q
100	100
104	

p = my_strrev()

p 100 q 100

p 104 q 100

my_strrev(100, 104-1)

→ my_strrev(100, 103);

q = p = p+1

p	q
100	100
104	100
105	105

Session – 57

→ Functions predefined string based functions

Session → 54

20-8-20

Functions → Predefined string based function

#include < string.h >

→ Function definition and declaration can't be changed for predefined functions.

(1) strlen()

declaration → size_t strlen(const char *s); ← unsigned int

strlen() returns unsigned int

strlen → calculate the length of string

```
#include <stdio.h>
#include <string.h>
void main()
{
  char s[20];
  Pf("Enter the string \n");
  Sf(" %s ", s);
  Pf(" %ld \n", strlen(s));
}
```

// Enter the string
hello
5

eg

Pf("%ld \n", strlen("abcd")); // 4

string constant gives the base address

Pf(" %ld \n", strlen("Hello"+1)); // 4 because here we pass address of 'e'

Pf(" %ld \n", strlen("Hello")+1);

Pf(" %c \n", "hello"[1]); // e

String constant & array name represents base address

Pf(" %s \n", "hello"[1]); // error

%s required address, but we are giving value

segmentation fault occurs

→ More than one argument will throw an error

eg:- ## strlen('a') // error seg fault (add a char const is passed)

→ for a string constant, when it is passed in the fun strlen(), it is treated as the base address.

2) strcpy() → copy a string

strncpy()

Prototypes

```
char *strcpy(char *dest, const char *src);
char *strncpy(char *dest, const char *src, size_t n);
```

→ If the programmer is not calculating the return value of any function, it is not an error.

```
#include<stdio.h>
#include<string.h>
void main()
{
  char s[20], d[20], *p;
  ptf("Enter string\n");
  sf("%s", s);
  p = strcpy(d, s);   ← the dest address is returned & it stored in p
  pf("s=%s d=%s p=%s\n", s, d, p);
}
```

src base add. ↓ | ↓ dest base addess | destination base address

o/p

```
Enter string
hello
s=hello d=hello p=hello
```

strncpy → copy only n char. from src to dest

strcpy - copy all char from src. to dest

strncpy:- **Note** → If there is no null byte among the first n byte of the src, the string placed in dest will not be, null terminated

→ If the length of the src is less than 'n' bytes, this func() will write addition null bytes to ensure that the total n bytes are written.

Session - 55 21-8-20

Functions, Predefined string based functions

strncpy - definition

```
char * strncpy(char * dest, const char *src, size_t n);
{                                can be src[i] only
    size_t i;
    for(i=0 ; i<n && src[i] != '\0' ; i++)
        dest[i] = src[i];
    for ( ; i<n ; i++)
        dest [i] = '\0';
    return dest;
}
```

3) strcmp() → strncmp()
Compare two strings

```
int strcmp(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, size_t n);
```

```
#include<string.h>
void main()
{
    char s[20], s1[20];
    pf("Enter the strings\n");
    sf(" %s %s", s, s1);
    pf(" %d\n", string(s, s1));
```

o/p
ab cd
am cd
-11
ASCII (18 ...)

string compare function returns
→ ASCII ref (in 64 bit OS) / Codec
→ 0, -1, 1 (in few gcc)
but if both strings are equal, return 'zero'

→ 64-bit C11 standard → returns ascii ref.
→ 32-bit C89 standard → returns 0, 1, -1

4) strcat() Concatenate two strings

```
char *strcat(char *dest, const char *src);
char *strncat(char *dest, const char *src, size_t n);
```

strcat defn:-

```
char *strncat(char *dest, const char *src, size_t n)
{
    size_t dest_len = strlen(dest);
    size_t;
    for(i=0; i<n && src[i]!='\0'; i++)
        dest[dest_len+i] = src[i];
        dest[dest_len+i] = '\0';
        return dest;
}
```

5) strchr/strrchr() → Locate character in string

```
char *strchr(const char *s, int c);
char *strrchr(const char *s, int c);
```

initially it is a character

→ returns the locations from the last, (returns the loc. of the last occurance of the char

strchr() returns the location of the occurence of the first char.

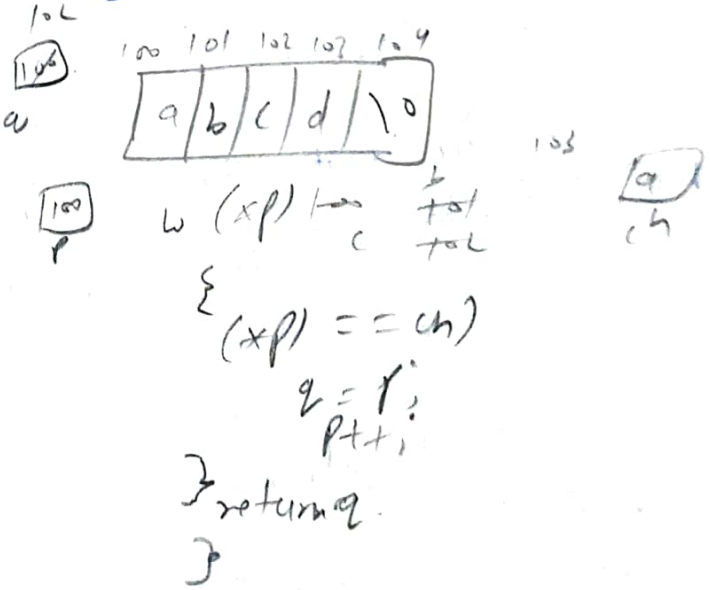
user-defined strchr()

```
char *my_strchr(char *, char);
void main()
{
    char s[20], ch, *p;
    Pf("Enter the string\n");
    sf("%s", s);
    Pf("Enter the char\n");
    sf("%c", &ch);
    pf("%p\n", s);
```

```
p= my_strchr(s,ch);
if(p==0)
printf("char not present \n");
else
printf("present at %p \n",p);
}
char *my_strchr(char *p, char ch);
{
  int i;
  i= my_strlen(p);
  for(i=i-1; i>=0; i--)
  { if(p[i]==ch)
      return p+i;
  }
  return 0;
}
```

using pointers

```
char *q =0; //making it a null pointer
while(*p)
{ if(*p==ch)
    q= p; // q will hold the last occurance of the character's address
  p++;
}return q;
}
```



Session - 5

Storage classes

24-03-20

→ auto
→ static } functions
→ extern
→ register

These 4 storage classes can be put before the datatypes

→ A storage class desides about these 4 aspects of a variable

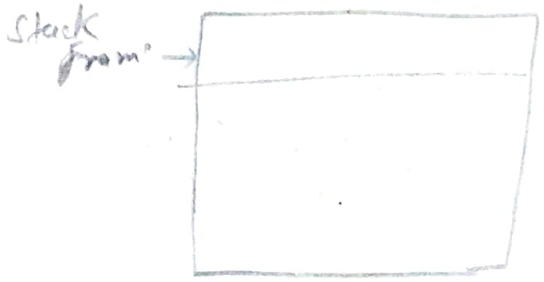
(1) Lifetime → time b/w the creation & destruction of the variable
(2) Scope (visibility) → Locations where the variable is available for use
(3) Initial value → Default value taken by an uninitialised variable
(4) Place of storage → Place in memory where the storage is allocated for the variable.

(1) Auto:-
→ If we declare a local variable without mentioning the storage type, compiler will default considers it as auto.
→ For an auto variable, the default value is an unexpected (garbage)
→ Auto variable can't be declared as global variable
→ All auto variable are local, but all local variable may not be auto.
→ All the auto variable are stored in function stack frame

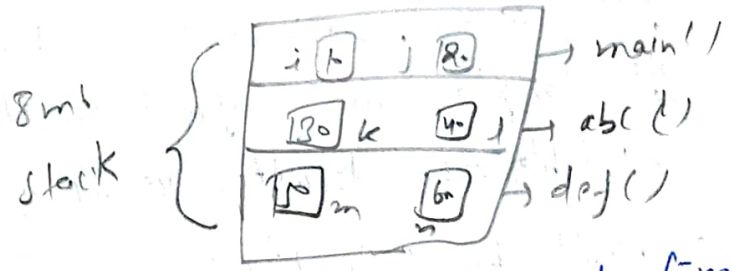
eg:-
```
#include<stdio.h>
void abc(void);
void def(void);
void main()
{
  int i=10, j=20;
  abc();
}
void abc(void)
{
  int k=30, l=40;
  def();
}
void def(void)
{
  int m=50, n=60;
}
```

Here main() → abc() → def()

8 mb of stack divided into stack frame



When stack calls the main(), main() stack frame is created.
In that stack i=10 & j=20 is stored.
Now, main() calls abc(), so abc() stack frame is created.
k=30, l=40 is stored here.
When abc() calls def(); def() stack frame is created
m=50, n=60 is stored.
When def() stack frame is created, main() & abc() frame still exist.
→ Stack frame hold the return addresses.



→ Size of stack frame is not fixed
When the functions are destroyed, its corresponding stacks frames are destroyed.

> Auto variables are stored in respected function's stack frame

→ Scope of an auto variable is within a blocks where we declared it.

eg:
```
void abc(void);
void main()
{
  int i;
  pf("i= %d \n", i);
}
void abc()
{
pf("i= %d \n", i); //
```

i is undeclared so error
'i' is declared in main() and is called in abc().

→ Life of an auto variable is starts when the stack frame is created and file ends when the stack frame is destroyed.

Example program:-

```
void abc(void);
void main()
{
  int i=10;
  abc();
}
void abc(void)
{
  int i=20;
  printf("i=%d\n",i);
}
```

Here life of abc() is created only when abc() is called.

Here, main()'s 'i' is destroyed first and then main()'s 'i' is destroyed at last.

This is in terms with stack's LIFO property (Last In First Out).

Session 57

25-8-20

```
int *abc(void);
void main()
{
  int *p;
  p=abc();
}
int *abc(void)
{
  int i=10;
  return &i;
}
```

In fun abc(), the 'i' decl. is an auto variable

When abc() completes execution, its stack frame is destroyed

But it returns the addr. of 'i' to main() & the stack where the memory for 'i' is there, is destroyed, so, the pointer may or may not contain the address of 'i', as its memory is destroyed.

Here, this pointer will be called as dangling pointer
Few compilers may throw an alert. It is not an error.
Here, the pointer is pointed to a memory where there's no life (variable reservation is ended)
~~to~~ the address of the auto variables should not be referred.

→ When we are designing any function, we should not return the address of auto variables, which we declared inside the function.
→ The pointer that tends to return the destroyed memory of an auto variable, is a dangling pointer
→ If we return the ~~address~~ address of an auto variable & if we catch it with a pointer, that pointer is called as dangling pointer.

```
dangling pointer
void abc(int *)
void main()
{
  int i=10;
}
```

2) Static variables

→ The default value of a static variable is '0'.

```
static int i=20;
void main()
{
  static int i=10;
  pf("i= %d\n", i);
}
```

o/p
i=0, this shows that local has the higher priority

Note → In C language, if one variable is declared globally as well as locally with the same name, local has the higher priority