ARRAY

C supports a derived data type known as array.

An array is a fixed size sequenced collection of elements of same data type.it is simply grouping of like type data

Example-> array name A of int data type

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 20 | 30 | 40 | 50 | 60 | 70 |

0 1 2 3 4 5

Types of array

1.one-dimensional array

2.two-dimensional array

3.three-dimensional array

ONE-DIMENSIONAL ARRAY

A list of item can be given one variable name using only one subscript and such a variable is called one -dimensional array.

Single subscript can begin with 0 and expressed as

X[0], x[1],x[2],x[3]……………..

Declaration of 1-d array

Data\_type variable\_name[size\_of \_array]

Eg, int height[10]

Reading values into array

For(i=0 ; i<10 ; i++)

{

Scanf(“%d”,&value);

}

Initialization of 1-d array

Data\_type name[size]={list of value};

Eg, int number[3]={1,2,3};

* If number of values is less then number of elements then remaining elements will be set to zero(0) automatically.

Eg, int a[5]={1,2,3}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 0 | 0 |

* The size can be omitted ,in such cases the compiler allocates enough spaces for all initialized elements.

Int counter[]={1,2,3,4,5};

Write a program to read n values in 1-d array and compute the sum of its square

#include<stdio.h>

Int main()

{

Int n;

Int sum=0;

Scanf(“%d”,&n);

Int a[n];

For(int i=0;i<n;i++)

{

Scanf(“%d”,&a[i]);

}

For(int j=0;j<n;j++)

{

Sum=sum+a[i]\*a[i];

}

Printf(“sum of square of array:%d”,sum);

Return 0;

}

1-d array and function

We can pass individual array elements as arguments

To a function like other simple variables.

Eg., function\_name(array\_name);

* We know changes made in formal arguments do not affect the actual arguments. but this is not the case of array .
* In case of simple variable the called function creates a copy of the variable and works on it,so any change made in the function do not effect the original values.
* When array is passed as an actual argument, the called function actually get access to the original array and works on it . so any changes made inside the function affect the original array.

eg., program to find the sum of square of all elements of array, show the new values of array element using function.

#include<stdio.h>

Void function(int val[]);

Int main()

{

Int arr[6]={1,2,3,4,5,6};

Function(arr);

Printf(“content of array are now”);

For(int i=0;i<6;i++)

{

Printf(“%d”,arr[i]);

}

Return 0;

}

Void function(int val[])

{

Int I,sum=0;

For(i=0;i<6;i+)

{

Val[i]=val[i]\*val[i];

Sum=sum+v[i];

}

Printf(“the sum of square=%d\n”,sum);

}

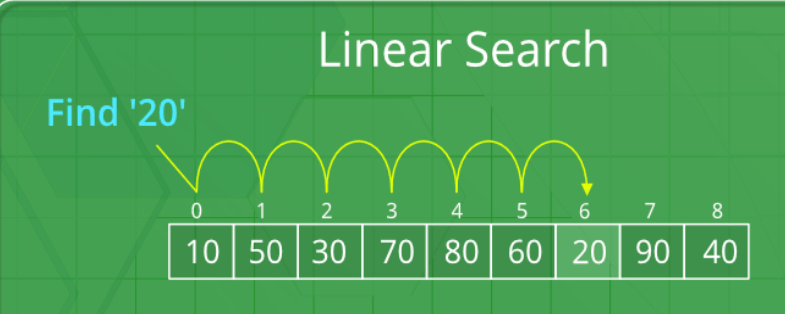
SEARCHING AND SORTING

Searching -> searching is the process of finding the location of the specified element in a list. The specified element is often called the search key.

The two most common used search techniques are:

* Sequential search
* Binary search

Sequential /linear search: Linear search is a sequential searching algorithm where we start from one end and check every element of the list until the desired element is found. It is the simplest searching algorithm.



Code:

#include <stdio.h>

int main()

{

int a[]={1,3,2,54,65,76};

int n=sizeof(a)/sizeof(a[0]);

int item,i;

printf("enter element to be searched:");

scanf("%d",&item);

for(i=0;i<n;i++)

{

if(item==a[i])

{

printf("%d found at %d",item,++i);

}

}

return 0;

}

Binary searching: A Binary Search is a sorting algorithm, that is used to search an element in a sorted array. A binary search technique works only on a sorted array, so an array must be sorted to apply binary search on the array. It is a searching technique that is better then the liner search technique as the number of iterations decreases in the binary search

Code: without using function

#include <stdio.h>

int main()

{

int a[]={1,2,3,54,65,76};

int n=sizeof(a)/sizeof(a[0]);

int item,i;

int last=n,first=0;

int mid=(last+first)/2;

printf("enter element to be searched:");

scanf("%d",&item);

while(first<=last)

{

if(a[mid]==item)

{

printf("%d is found at %d\n",item,++mid);

break;

}

if(a[mid]<item)

first=mid+1;

if(a[mid]>item)

last=mid-1;

mid=(last+first)/2;

}

return 0;

}

Code: using function

#include <stdio.h>

int search(int arr[],int s,int n)

{

int last=n,first=0;

int mid=(last+first)/2;

while(first<=last)

{

if(arr[mid]==s){

return (mid+1);

}

if(arr[mid]<s)

first=mid+1;

if(arr[mid]>s)

last=mid-1;

mid=(last+first)/2;

}

return 0;

}

int main()

{

int pos;

int a[]={1,2,3,54,65,76};

int n=sizeof(a)/sizeof(a[0]);

int item,i;

printf("enter element to be searched:");

scanf("%d",&item);

pos=search(a,item,n);

if(pos==0)

{

printf("%d not found",item);

}else

{

printf("%d is found at %d",item,pos);

}

return 0;

}

Sorting->sorting is the process of arranging element in the list according to their values ,in ascending or descending order .

Types of sorting :

1.bubble sort

2.selection sort

3.insertion sort

4.quick sort

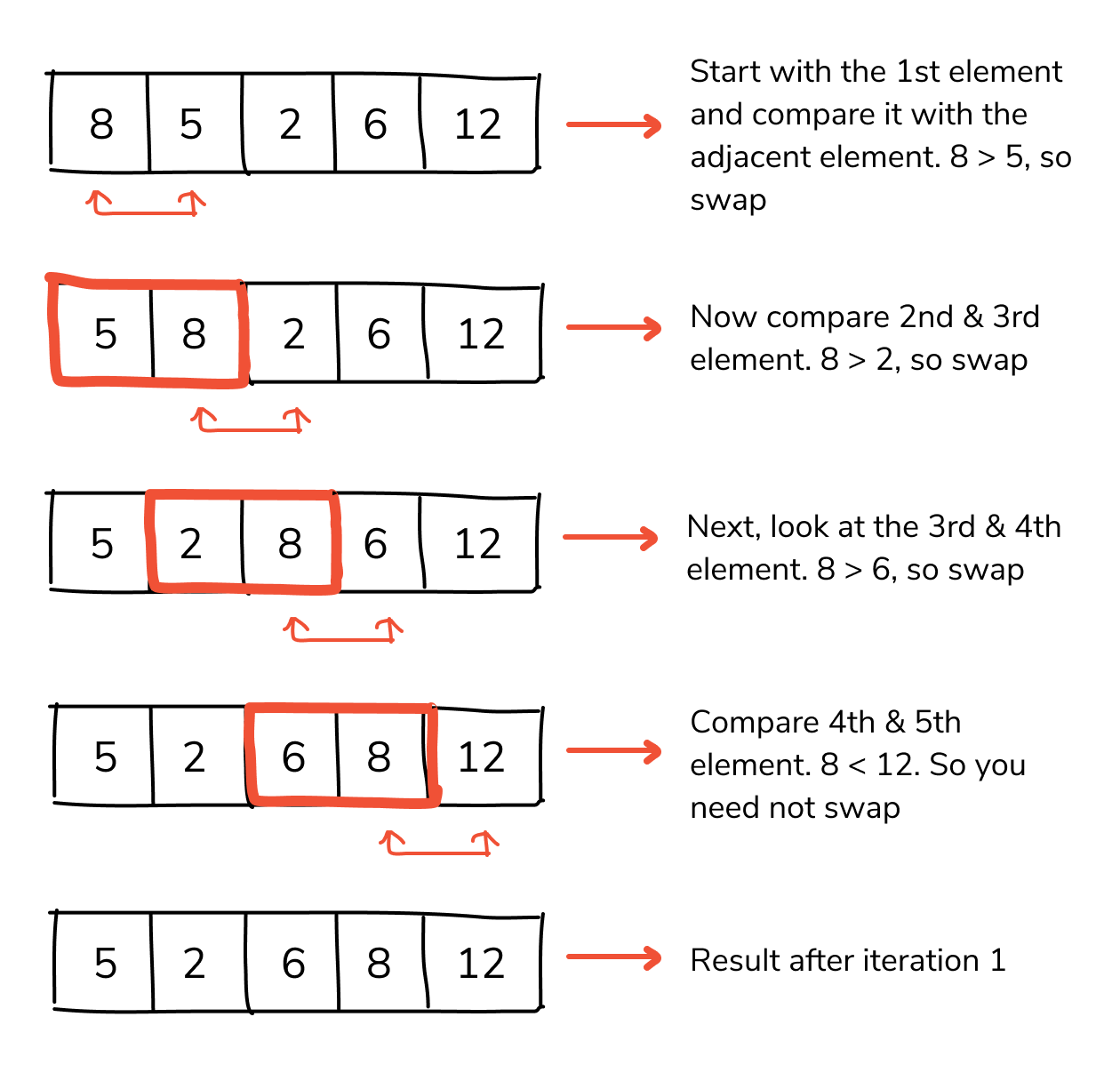
5.merge sort

6.Shell sort

**BUBBLE SORT**

**Link:** **https://youtu.be/bwadL7uIFiU**

bubble sort is also known as sinking sort. this algorithm compares each pair of adjacent items and swap them if they are in the wrong order.



Code: for ascending order

#include <stdio.h>

int main()

{

int i,j,temp,n;

printf("enter max number of sort");

scanf("%d",&n);

int a[n];

printf("enter %d element :",n);

for(i=0;i<n;i++)

{

scanf("%d",&a[i]);

}

for(i=0;i<n;i++)

{

for(j=i+1;j<n;j++)

{

if(a[i]>a[j]) //for descending order if(a[i]<a[j]

{

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

}

for(i=0;i<n;i++)

{

printf("%d,",a[i]);

}

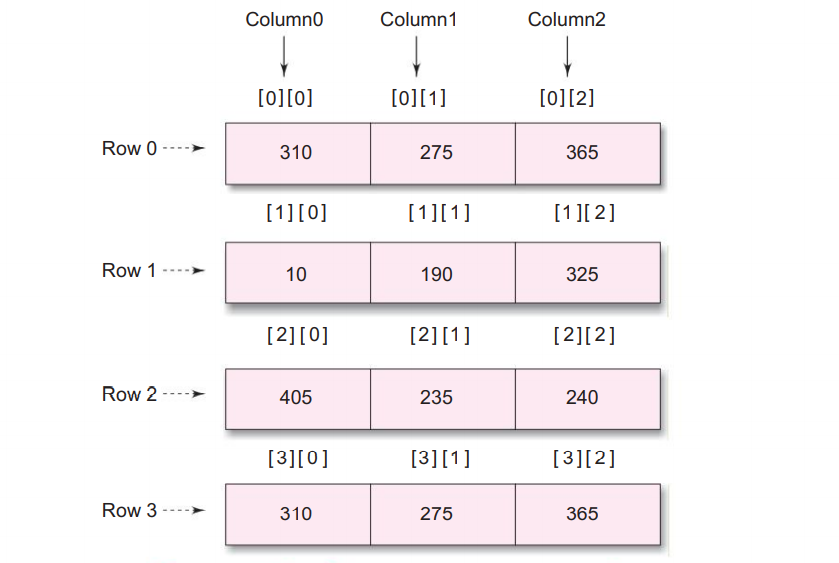
return 0;

}

Two-dimensional array

DECLRATION Syntax:

Data\_type array\_name[row\_size][column\_size]



INITIALIZITION SYNTAX

int table[2][3] = { 0,0,0,1,1,1};

* the initialization is done by row by row.

Int table[2][3]={{0,0,0},{1,2,3}};

* we can also initialize 2-d array

int table[2][3]={

{0,0,0},

{1,2,3}};

MULTI DIMENSIONAL ARRAY

c allows array of three or more dimension. The exact limit is determine by the compiler

* general syntax is given below:

data\_type array\_name[s1][s2][s3]…..[sn]

PASSING ARRAY TO FUNCTION:

One dimensional:it is possible to pass the values of an array to a function .it is sufficient to list the name of the array,without any subscripts and size of array,

#include<stdio.h>

Int add(int a[]);

Int main()

{

Int arr[5]={1,2,3,4,5};

add(arr);

}

When dealing with array arguments we should remember one major distinction. if a function changes the values of the elements of an array ,then these changes will be made to the original array that passed to the function.

Two-dimensional array

We can also pass multi-dimensional arrays to functions.

* The function must be called by passing only the array name.
* In the function definition we must indicate that array has two-dimension by including two sets of brackets.
* The size of the second dimension must be specified.

#include<stdio.h>

Int add(int a[][n]);

Int main()

{

Int arr[2][3]={{1,2},{3,4},{5,6}};

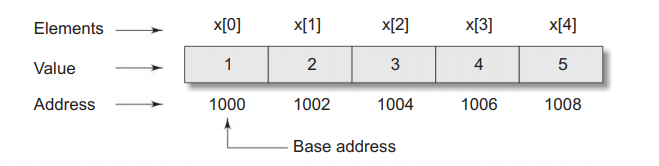
add(arr);

}

POINTERS AND ARRAYS

When an array is declared ,the compiler allocates a base address and sufficient amount of storage to contain all the elements of the array in contiguous memory location.

BASE ADDRESS is the the location of the first element of the array



The name x is defined as the pointer pointing to the first element-> x[0]

x=&x[0]=1000

if we declare p as an integer pointer then we can make the pointer p to point to the array x by the following assignment.

P=x;//initializing with base address

P=&x[0];

Now we can access every value of x using p++ to move from one element to another.

P=&x[0];=(1000)

P++ P+1=&x[1];(1002)

P++ p+2=&x[2];=(1004)

P++ p+3=&[3];=(1006)

P++ p+4=&[4];=(1008)

Now for values:

\*p=x[0];=1

\*(p+1)=x[1];=2

\*(p+2)=x[2];=3

\*(p+3)=x[3];=4

\*(p+4)=x[4];=5

Program to print the value and the address of each index in an array.

#include <stdio.h>

int main()

{

int \*p;

int i;

int a[5]={1,2,3,4,5};

i=0;

p=a;

for(i=0;i<5;i++,p++)

{

printf("value=%d address=%p \n",\*p,p);

}

return 0;

}

Program to find the sum of array using pointer.

#include<stdio.h>  
int main()

{

int \*p;

int i,sum=0;

int a[5]={1,2,3,4,5};

i=0;

p=a;

for(i=0;i<5;i++,p++)

{

Sum=sum+ \*p;

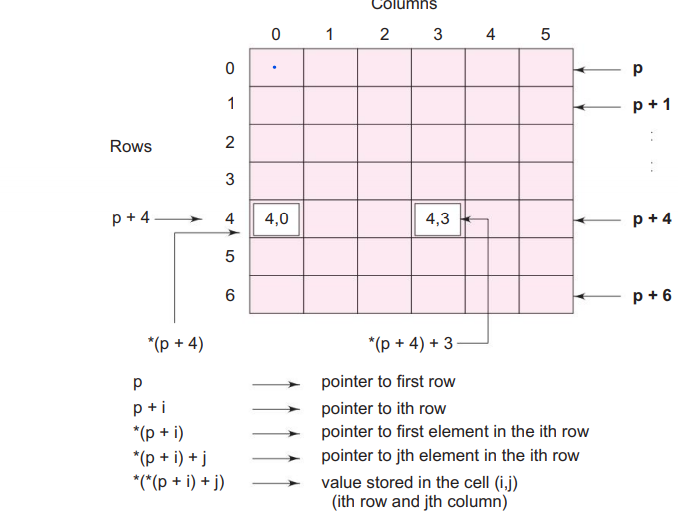
}

Printf(“sum=%d”,sum);

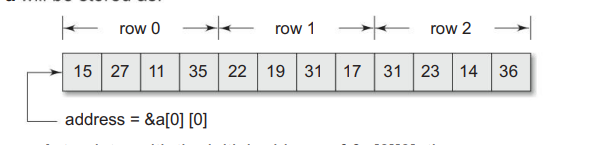
return 0;

}

* Pointer can be used to manipulate two -dimensioarrays as well.



The base address of the array “a” is &a[0][0] and starting at this address, the compiler allocates contiguous space for all the element row wise .



**STRING**

a string is a sequence of characters that is treated as a single data item

eg. “man”

Character strings are often used to build meaningful and readable programs. The common operations performed on character strings include:

\* Combining strings together.

\* Copying one string to another.

\*Comparing strings for equality.

\*Extracting a portion of a string

DECLARING

C does not support string as a datatype .but allow us to represent strings as character arrays

SYNTAX: char string\_name[size];

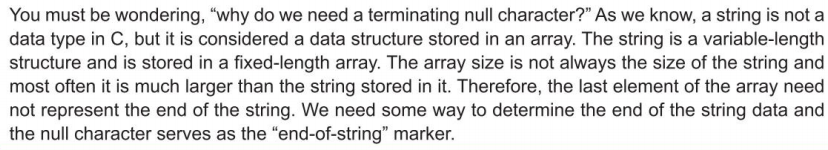
When compiler assigns a character string to character array , It automatically supplies a null character (‘\0’) at the end of the string .therefore the size should be equal to the equal to the maximum number of character string plus one.

INITIALIZING STRING

Char city[9]= “new York”;

Or

Char city[9]={‘n’,’e’,’w’,’ ‘,’y’,’o’,’r’,k’,’\0’};



READING STRINGS FROMM TERMINAL

* Using scanf

Char address[10];

Scanf(“%s”,address);

But problem with scanf is that it terminates its input on the first whitw space it finds.

* Other method using scanf: c supports a format specification known as the edit set conversion code %[….].

Char line[80];

Scanf(“%[^\n]”,line);

* Using getchar and gets function

GETCHAR:we can use the function to read a single character from the terminal.

Chr ch;

Ch=getchar();

* Getchar function has no parameters.

GETS()

Another more convenient method of reading a string of text containing whitespaces is to use the library function gets available in <stdio.h> header file.

SYNTAX : gets(str);

Str is string for a instance.

* Unlike scanf ,ot does not skip whitespaces.
* (Be careful not to input more character that can be stored in the string variable used. Since C does not check array-bounds, it may cause problems.)

PRINTING STRING

We use the printf function with %s format specifier .

Printf(“%s”,str);

s

USING PUTCHAR AND PUTS FUCTION

* PUTCHAR:like getchar ,c supports another character handling function putchar

To output the values of character variables

E.G., char ch = ‘a’;

Putchar(ch);

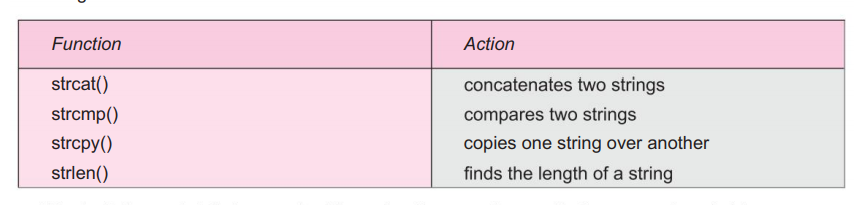
The function putchar requires one para meter.

* PUTS: another way of printing string values is to use the function “puts” decalared in the header file<stdio.h>.this is one parameter function and invoked as under.

Puts(str);

STRING HANDLING FUNCTION

C supports a large number of string handling functions that can be used to carry out many of the string manipulations

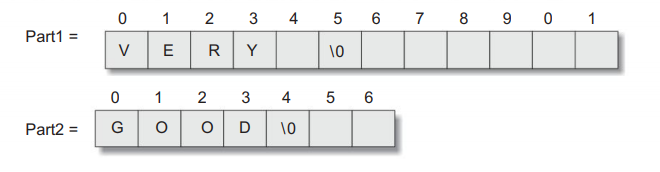


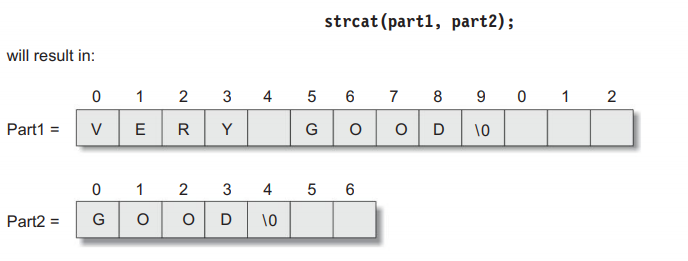
STRCAT() function

The strcat() function joins two strings together .

Syntax: strcat(string1,string2);

String1 and string2 are character arrays. When strcat function is executed string2 is appended to string1, it does so by removing the null character at the end placing string1 and string2 from there.string2 remain unchanged.





We must make sure that the size of string is large enough to accommodate the final string.

C PERMITS NESTING OF strcat function.

Strcat(strcat(string1,string2),string3);

STRCMP() function:the strcmp function compares two string identified by the arguments and has a value 0 if they are equal ,and if they are not it has the numeric difference between the first non-matching charactes in the string

Syntax: strcmp(string1,string2);

STRCPY() function:

The function worls almost like string-assignement operator. It takes the form:

Syntax: strcpt(string1,string2);

This function will assign the contents of the string2 to the string1. Also size of string1 should be large enough to receive the contests of string2.

STRLEN() function:

This function counts and return the number of characters in a string.it takes the form

Syntax: n=strlen();

The counting ends at the null character.

OTHER STRING FUNCTION:

The header file <string.h> contains many more string manipulation they might be useful in certain situations.

STRNCPY()

Strncpy copies only the left-most “n” characters of the source to the target string variable. this is a three -parameter fuction

Syntax:strncpy(s1,s2,n);

This statement copies the first characters of the source string s2 into the target string s1.

* Since the first 5 characters may not include the terminating null character ,we have to place it explicitly inn the 6th position of s2

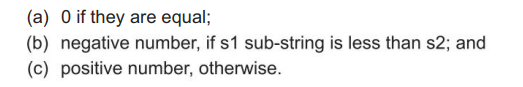
S1[6]= ‘\0’;

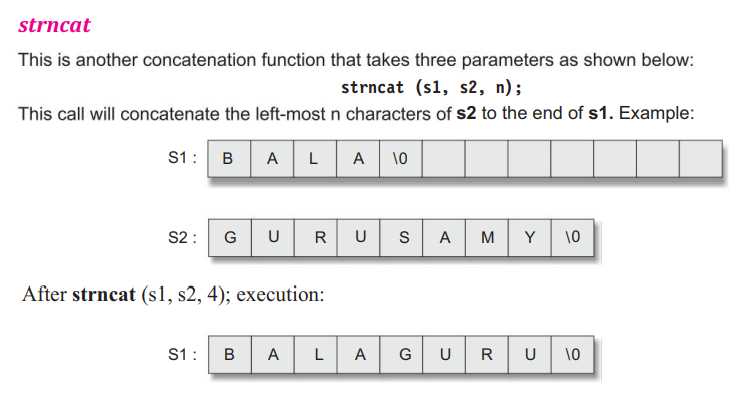
STRNCMP()

A variation of the function strcmp is the function strncmp. This function has three parameter

Syntax: strncmp(s1,s2,n);

This compares the leftmost n characters of s1 to s2 and returns:





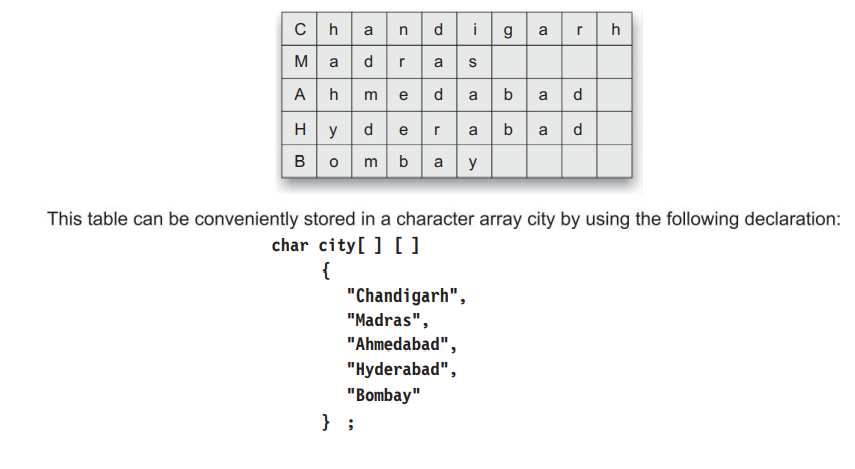
STRSTR: it is two-parameter function that can be used to locate a sub-string in a string.

Syntax: strstr(s1,s2);

The function strstr searches the string s1 to see wheather the string s2 is contained in s1.if yes the function returns the position of the first occurrence of the sub-string ,otherwise, it returns a NULL pointer;

TABLE OF STRINGS

A list can be treated as a table of strings and a two-dimensional character array can be used to store the entire list.



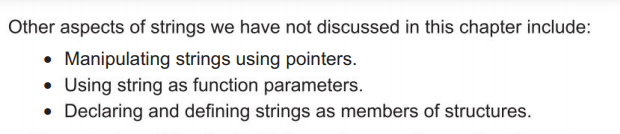
To access the name of the its city in the list ,we write

City[i-1]

And therefore city[0] denotes chandigarh;

City[1] denotes madras

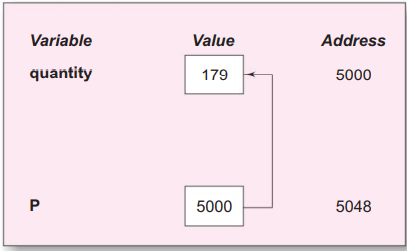
Note that a two-dimensional array is used to store the list of strings. Each strings is read using a scanf function with %s format .also if any string contains a white space, then the part the white space will be treated as another item I the list by the scanf.



Pointer

Pointer contain memory address as their

values



Pointer

Since pointer is variable its value is also stored in the memory in another location

* Pointers are more efficient in handling arrays and data tables.
* Pointers can be used to return multiple values from a function via function arguments.
* Pointer allow c to support dynamic memory management.
* They increase the execution speed .

The & operator immediately preceding a variable returns the address of the variable associated with it.

P=&quantity ;

Declaring pointer variable

Data\_type\*pt\_name;

* The asterisk (\*) tells that the variables pt\_name is a pointer variable.
* Pt\_name needs a memeory location.
* Pt\_name points to variable of type data\_type.

e.g int \*p;

declares the variable p as a pointer variable that points to an integer data type.

Int x,y;

Int \*p;

X=10;//assigning 10 to x

P=&x;// assigning address of x to p

Y=\*p;// accessing x through p

\*p=10;//assigning 20 to x

Initialization of pointer variables

Int a;

Int\*p;//declaration

P=&a;//initialization

Rule:the only requirement is that the variable ‘a’ must be declared before the initialization takes place

< this the inintialization of p and not \*p>

* With the exception of NULL and 0, no other constant value can be assigned to a pointer variable.

ACCESSING A VARIABLE THROUGH POINTER

This is done by using another unary operator \*(asterisk),usually known as the indirection operator.

Int q,\*p,n;//declaring ‘q’and ‘n’ as integer and p as pointer

q=179;//assign value 179 to q

p=&q;//assign address of q to p

n=\*p;//when ‘\*’ is placed before a pointer variable in an expression the pointer returns the value of the variable which it contains the address. the \* can be remembered as value at address

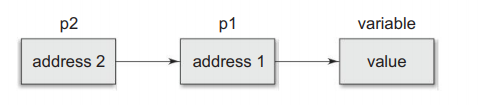
so the value of n would be 179.

P=&q; n=\*&q; n=q;

N=\*p;

CHAIN OF POINTER

It is possible to make a pointer to point to another pointer, thus creating a chain of pointers as shown.



Here, the pointer variable p2 contains the address of the pointer variable p1, which points to the location that contains the desired value. this is known as multiple indirections .

A variable that is a pointer to a pointer must be declared using additional indirection operator symbols.

Int \*\*p2; 🡪 declaration example🡨

POINTER EXPRESSION

Y=\*p1 +\*p2; //if want to and the value at address p1 and p2 is holding//

Y=p1+p2 //illegal

* C allows us to add integers to or subtracts integers from pointers as well as to subtract one pointer from another pointer
* then p2 – p1 gives the number of elements between p1 and p2.
* We may not use pointers in division or multiplication
* Eg. P1/p2or p1\* p2or p1/3

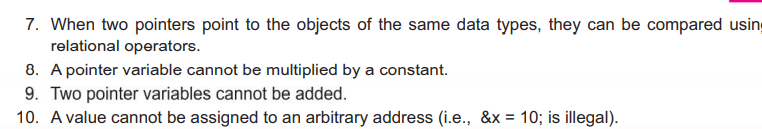
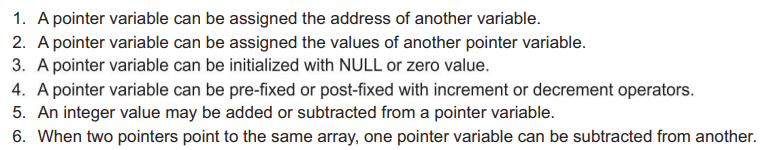
POINTERS INCREMENTS

P1++;

Will cause the pointer p1 to point to the next value of its type.

For example p1 contains the value 2800

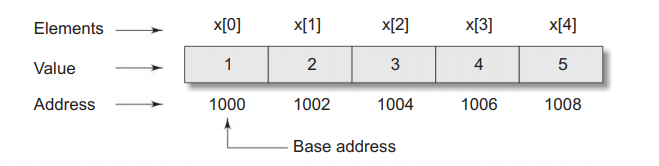
And after p1++ the value of p1 will be 2802 not 2801.



POINTERS AND ARRAYS

When an array is declared ,the compiler allocates a base address and sufficient amount of storage to contain all the elements of the array in contiguous memory location.

BASE ADDRESS is the the location of the first element of the array



The name x is defined as the pointer pointing to the first element-> x[0]

x=&x[0]=1000

if we declare p as an integer pointer then we can make the pointer p to point to the array x by the following assignment.

P=x;//initializing with base address

P=&x[0];

Now we can access every value of x using p++ to move from one element to another.

P=&x[0];=(1000)

P++ P+1=&x[1];(1002)

P++ p+2=&x[2];=(1004)

P++ p+3=&[3];=(1006)

P++ p+4=&[4];=(1008)

Now for values:

\*p=x[0];=1

\*(p+1)=x[1];=2

\*(p+2)=x[2];=3

\*(p+3)=x[3];=4

\*(p+4)=x[4];=5

Program to print the value and the address of each index in an array.

#include <stdio.h>

int main()

{

int \*p;

int i;

int a[5]={1,2,3,4,5};

i=0;

p=a;

for(i=0;i<5;i++,p++)

{

printf("value=%d address=%p \n",\*p,p);

}

return 0;

}

Program to find the sum of array using pointer.

#include<stdio.h>  
int main()

{

int \*p;

int i,sum=0;

int a[5]={1,2,3,4,5};

i=0;

p=a;

for(i=0;i<5;i++,p++)

{

Sum=sum+ \*p;

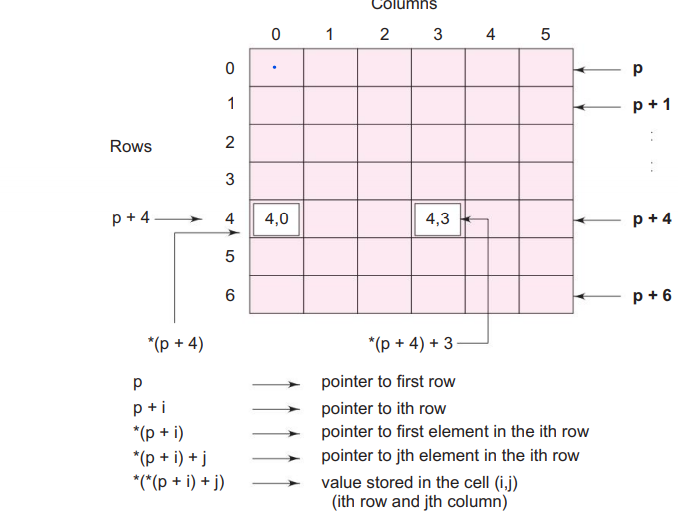
}

Printf(“sum=%d”,sum);

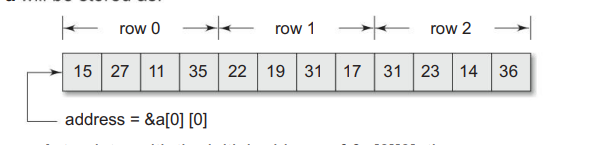
return 0;

}

* Pointer can be used to manipulate two -dimensioarrays as well.



The base address of the array “a” is &a[0][0] and starting at this address, the compiler allocates contiguous space for all the element row wise .



ARRAY OF POINTERS/strings and pointer

One important use of pointers is in handling of a table of strings

Char name[3][25];

This says that the name is a table table containing three names ,each with a maximum length of 25 characters. The total storage requirements for the name table 75 bytes.

* We know that rarely the individual strings will be equal lengths so instead of making each row a fixed number of characters we can make it a pointer to a string of varying length
* Eg

Char \*name[2]={

“new Zealand”,

“australia”,

};

To print out all three names:

For(i=0;i<2;i++)

Printf(“%s\n”,name[i]);

* To accesss the jth character in the ith name

\*(name[i]+j);

POINTER AS FUNCTION ARGUMENTS

USER DEFINED FUNCTION

Function is one of the strength of c language

C function can be classified into two categories namely library function and user-defined function. Main is the example of user defined function and scanf and printf belong to the category of library function.

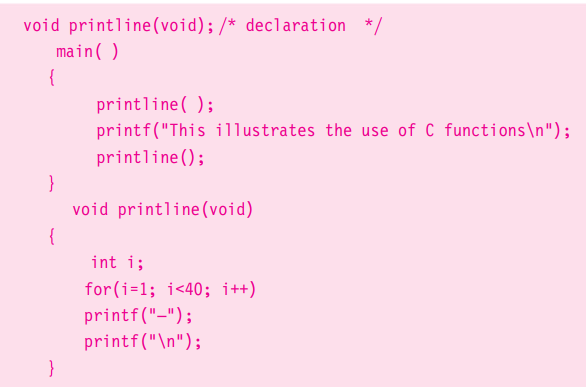
As said e earlier ,main is specially recognized function in c. every program must have a main function to indicate where the program has to begin its execution

The program may become too large and complex and as result the task of debugging ,testing and maintaining becomes difficult. If a program is divided into functional parts, then each part may be independently coded program called sub program. Such c program is called function

Advantages of function

-the length of source program can be reduced by using functions at appropriate places.

-it is easy to locate and isolate a faulty function



ELEMENTS OF USER-DEFINED FUNCTION

There are three elements that are related to function:

-function definition: it is an independent program module that is specially written to implement the requirements of the function.

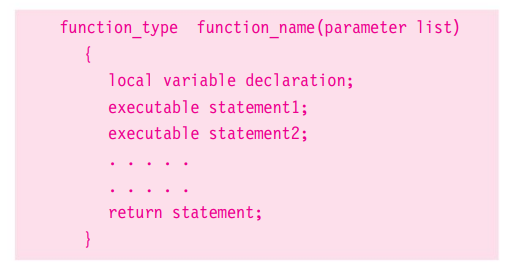
-function call : in order to use this function we need to invoke it at required place in the program. This is known as the function call.

-function declaration: the calling program should declare any function that is to be used later in the program. This is known as the function declaration or function prototype.

FUNCTION DEFINATION

A function definition shall include the following elements

1. Function name
2. Function type FUNCTION HEADER
3. List of parameters
4. Local variables
5. Function statements FUNCTION BODY
6. As return statements



Function body

Function header

-the function type specifies the type of value that function Is expected to return. if it is not specified then it is int type if function is not returning anything then we need to specify return type as void.

- function name is any valid c identifier

-the parameter list declares the variable that will receive the data sent by calling program. they serve as a input data in a function to carry out the specified task. they are often referred to as FORMAL PARAMETERS

-FUNCTION BODY: the function body contains the declaration and statements necessary for performing the required task.

The body enclosed in braces ,contains three parts

-local declarations that specify the variables needed by the function

-function statements that performs the task of the function

-a return statement that returns the value evaluated by the function

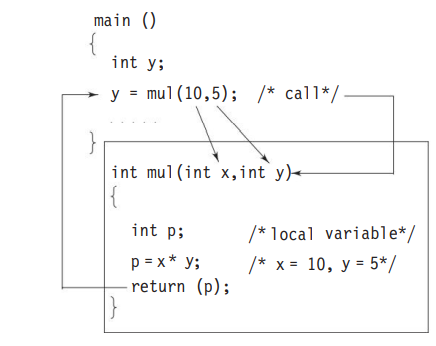
RETURN VALUES AND THEIR TYPES

A function may or may not send back any value to the calling function,

* It is possible to pass to the called function any number of values, the called function can only return one value per call at the most

FUNCTION CALLS

A function can be called by simple using the function name flowed by the list of actual parameter enclosed in parentheses



When the compiler encounters a function call, the control is transferred to the function and get back to main when return statement is encountered.

FUNCTION DECLARATION

Like variables, all function in a c program must be declared, before they are invoked. A function declaration consists of four parts

* Function type (return type)
* Function name
* Parameter list
* Terminating semicolon



A prototype declaration may be placed in two places in a program

1. Above all the functions
2. Inside a function definition

* When we place the declaration above all the functions, the prototype is referred to as a global protype. Such declaration are available for all the functions in the program.
* When we place it in a function definition, the prototype is called a local prototype. such declarations are primarily used by the function containing them.
* The place of declaration of a function defines a region in which the function may be used by other functions. This region is known as the scope of the function.

PARAMETERS EVERYWHERE

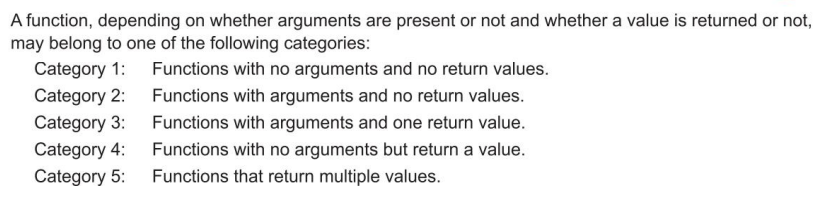
Parameters (also known as arguments) are used in three places:

1. In declaration
2. In function
3. In function

The parameters used in prototypes and function definitions are called formal arguments and those used in function calls are called actual parameter.

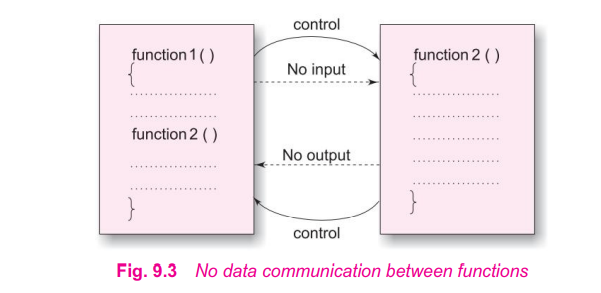
Actual parameters can be simple constant, variable, or expressions.

CATEGORY OF FUNCTIONS



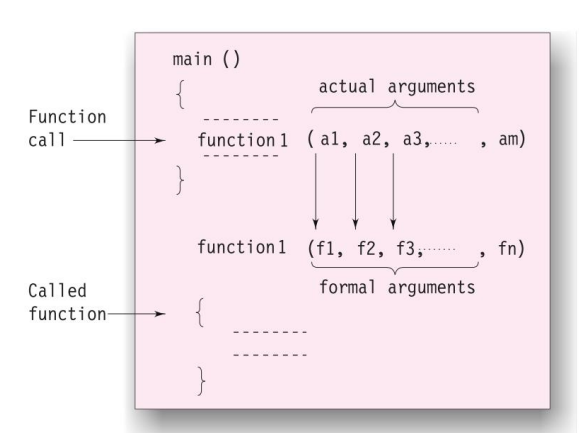
* No arguments and no return values

When a function has no arguments, it does not receive any data from the calling function. Similarly when it does not return a value , the calling function does not receive any data from the called function.



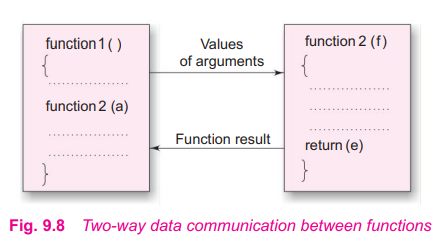
* Arguments but no return values

The nature of data communication between the calling function and the called function with arguments but no return value



* Arguments with return values

The function value receives data from the calling function through arguments, but does not send back any value. Rather, it displays the results of calculations at the terminal.



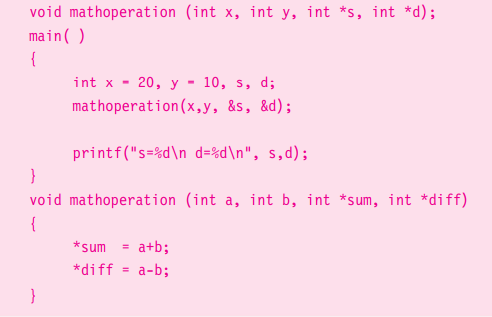
* No arguments but returns a value

There could be occasions where we may need to design functions that may not take any arguments but returns a value to the calling function.

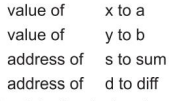
For example: the GTECHAR function has no parameters but it returns an integer type data that represents a character.

* Functions that return multiple values

The mechanism of sending back information through arguments is achieved using what are known as the address operator(&) and indirection(\*) operator



The actual arguments x and y are inputs arguments ,s and d are output arguments. In the function call, we pass the address of locations where the values of s and d are stored in the memory

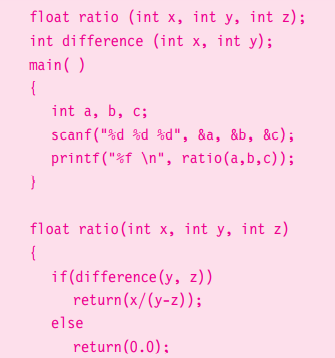


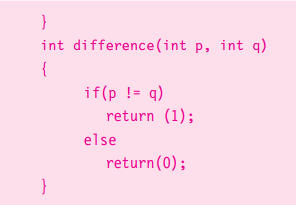


NESTING OF FUNCTIONS

C permits nesting of functions , freely . main calls function1, function1 calls function2 , which calls function3,……… and so on.

There is no limits

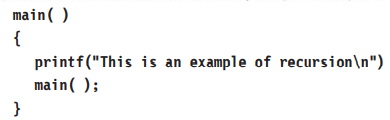




RECURSION

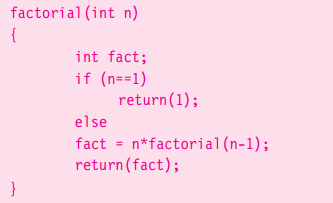
Recursion is where a function calls itself .

Eg.



This program execution will continue indefinitely

Another useful example of recursion is the evaluation of factorial of a given number.

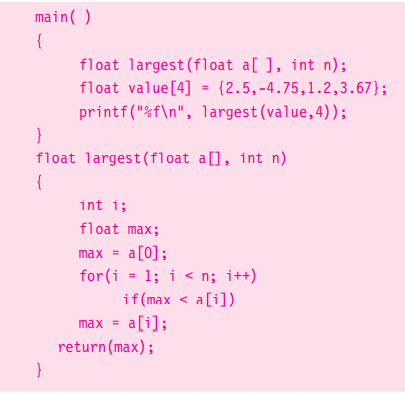


We must have IF statement somewhere to force the function to return without the recursive call being executed .otherwise the function will never returned.

PASSING ARRAYS TO FUNCTIONS

* One-dimensional arrays

It is also possible to pass the values of an array to a function. To pass a one-dimensional array to a called function , it is sufficient to list the name of the array without any subscript.

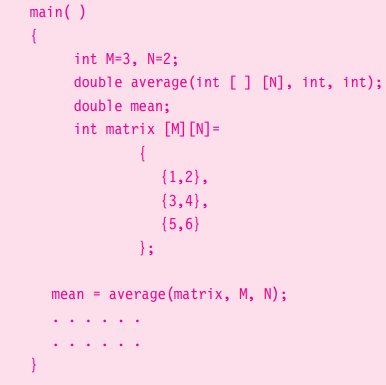


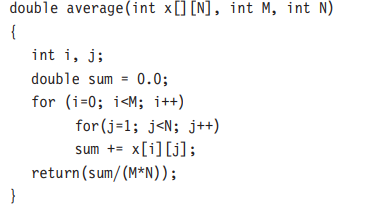
In c , the name of the array represents the address of its first element. By passing the array name, we are in fact passing the address of the array to the called function, therefore any changes in the array in the called function will be reflected in the original array.

TWO-DIMENSIONAL ARRAYS

We can also pass multi-dimensional arrays to functions. The approach is similar to the one we did with 1-dimensional

* The function must be called by passing only the array name
* In the function definition, we must indicate that the array has two-dimensional by including two sets of brackets
* The size of the second dimension must be specified.
* The prototype declaration should be similar to function header.





PASSING STRINGS TO FUNCTIONS

The strings are treated as character arrays in c and therefore the rules for passing strings to functions are very similar to those for passing arrays to functions

Basic rules are:

* The string to be passed must be declared as a formal argument of the function when it is defined

Void display( char array[])

{

………………………………

………………………………

}

* The function prototype must be declared must show that the argument is string.

Void display( char str[])

* A call to the function must have a string array name without subscripts as its actual arguments e.g

Display(names);

**PASS BY VALUE VS PASS BY ADDRESS/POINTERS**

The technique used to pass data from one function to another is known as parameter passing.

Parameter passing is done in two ways:

* **Pass by value (also known as call by value):** in pass by value, values of actual parameters are copied to the variables in the parameter list of the called function. The called function works on the copy and not on the original values of the actual parameters. This ensures that the original data in the calling function cannot be changed accidentally.
* **Pass by pointers (also known as pass by pointers):** the memory addresses of the variables rather than the copies of values are sent to the changed values are available in the calling function for its use.

THE SCOPE, VISBILITY AND LIFETIME OF VARIABLES:

In c not only do all variables have datatypes , they also have a storage classes are most relevant to functions

1.automatic variable

2. external variables

3. static variables

4. register variables

*Scope of variable determines over what region of the program a variable is actually available for use (****ACTIVE****).*

*Longevity /lifetime refers to the period during which a variables retains a given value during execution of program (****ALIVE****).*

The visbility refers to the accessibility of a variable from the memory.

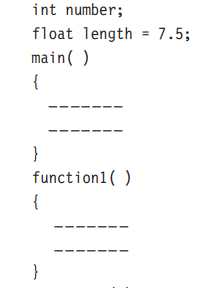
The variables may also broadly categorized depending on the place of their declaration as internal(local) and external (global).

* **AUTOMATIC VARIABLES:** these variables age declared inside the function in which are used to be used
* they are created when function is called and destroyed automatically when the function is exited.
* These are local or internal variables.

1. Lifetime: any variable variable local to main will be normally **alive** throughtout the whole program
2. **Scope:** it is active only in main.

* **EXTERNAL VARIABLES:** variables that are both **alive** and **active** throughtout the entire program are known as external / global variables.

external variables are outside a function.



* In case a local variable have the same name , the local variable will have precedence over the global one in the function declared.
* If global variable is declared after main function the main cannot access the variables this problem can be solved by declaring the variable with the storage class **extern**

Main()

{

Extern int y ; // external declaration

}

Func()

{

Extern int x; // external declaration

}

**STATIC VARIABLES:** the value of static variable persist until the end of the program. a variable can be declared static using the keyword static like.

* Static variable may be either on internal type or an external type depending on the place of declaration.
* Internal static variables are those which are declared inside a function.
* An external static variables is declared outside of all function and is available to all the function in that program.
* The difference between a static external variable and a simple external variable is that the static external variable is available only within the file where it is defined while the simple external variable can be accessed by other files.
* **REGISTER VARIABLES:**  we can tell the compiler that a variable should be kept in one of the machine’s registers, instead of keeping in the memory. since a registers. Instead of keeping in the memory . since a register access is much faster than a memory access, keeping the frequently accessed variables in the register will lead to faster execution of programs.

**Register int count;**

**UNDERSTANDING ACTIVATION RECORD/STACK FARME**

#include<stdio.h>

Void result(int value)

{

Printf(“factorial is %d \n”,value);

}

Void factorial(int num)

{

Int fact;

For(fact=1;num>1;num--);

Result (fact);

}

Int main()

{

Int i=5;

Factorial(i);

}

|  |  |
| --- | --- |
| **ACTIVATION RECORD OF RESULT ()**  **Local variable:value** | **Thirdly result is called then under result the**  **The local variable is called** |
| **ACTIVATION RECORD OF FACTORIAL()**  **LOCAL VARIABLE: fact,num,result(fact)** | **Second factorial() is called then under factorial(), result is called** |
| **ACTIVATION RECORD OF MAIN()**  **LOCAL VARIABLE:I, factorial(i)** | **First main() is called then under main() , factorial() is called.** |

* **Stack frame can be visualized as file of dishes as a buffet**
* **Order of removal is last in first out LIFO**
* **Adding items is reffered to as PUSHING**
* **Removing items is reffered to as POPPING**
* **Function call/return mechanism is modelled as on the stack data structures**
* **Let us say function A calls B ,B calls C , and C calls D and so on.**
* **We must have a way of coming back to the caller each time.**
* **On each call of a function a data frame is pushed onto the stack.**
* **This item is known as stack frame os the activation record.**
* **Activation record contains return address of the caller .**
* **Function on completing execution causes stack frame to be popped and control goes back to the caller using return address in the popped stack frame.**
* **Finite amount of memory is allotted for the call stack .if function are called endlessly then the stack overflows.**
* **Local variables of the called function and parameters to the called function are on the stack frame of that function**
* **That is why when you exit a function call the local variables are not longer accessible since they are on the stack frame which been popped out.**
* **ACTIVATION RECORD is active only when the function is in the STATE OF EXECUTION till a return or end of the function code.**