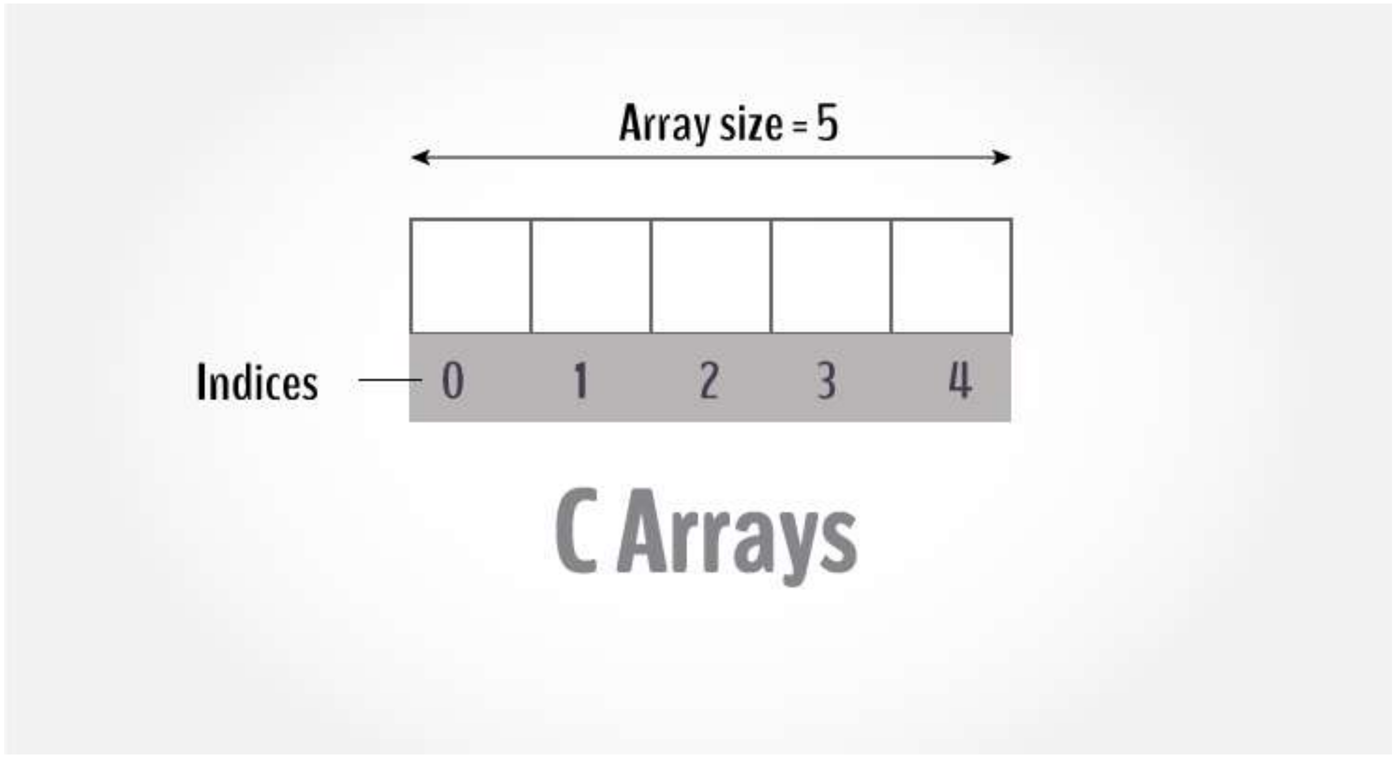
UNIT IV

Arrays

- An array is a variable that can store multiple values of same data type.

int marks[5]

marks→



How to declare an array?

- For example, if you want to store marks obtained in C for 70 students(70 integer values), you can create an array for it.

How to declare an array?

dataType arrayName[arraySize];

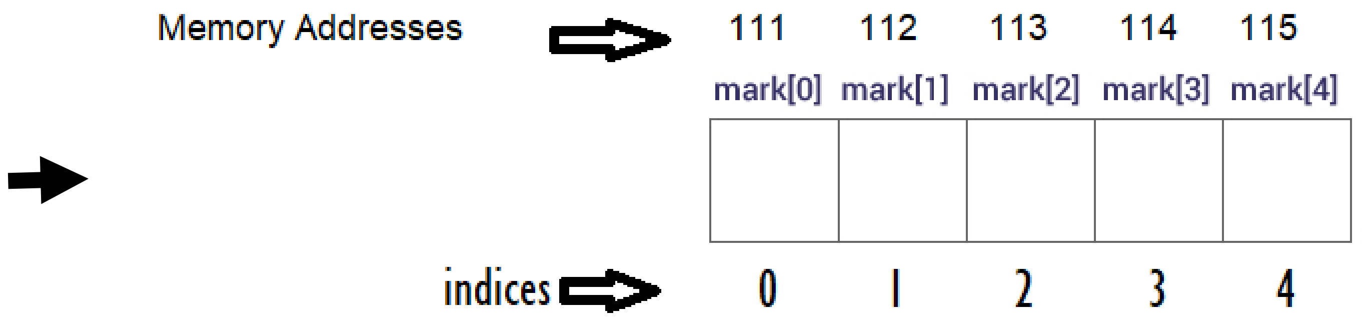
int marks[70];

Here, we declared an array, marks, of integer data type. And its size is 70. Meaning, it can hold 70 integer values.

- You can access elements of an array by indices.
- The first element is mark[0], the second element is mark[1] and so on.

Few keynotes:

- mark



Few keynotes:

•Arrays have 0 as the first index, not 1. In this example, mark[0] is the first element.

•If the size of an array is n, to access the last element, the n-1 index is used. In this example, mark[4].

•Also, the array name stores the memory address of the first element

mark=111

How to initialize an array?

- It is possible to initialize an array during declaration. For example,

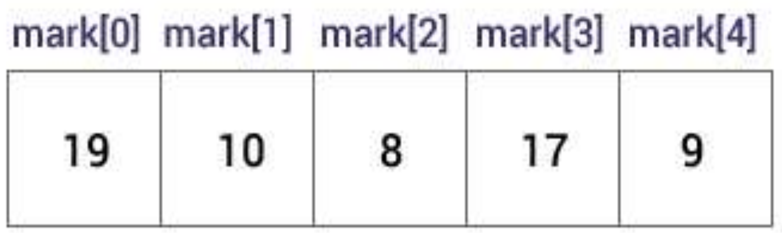
    int mark[5] = {19, 10, 8, 17, 9}; mark[0]=19 mark[1]=10 mark[2]=8 and so on

- You can also initialize an array like this.

```
int mark[] = {19, 10, 8, 17, 9};
```

- Here, we haven't specified the size. However, the compiler knows its size is 5 as we are initializing it with 5 elements.

Few keynotes: int mark[5];



- mark[0] is equal to 19
- mark[1] is equal to 10
- mark[2] is equal to 8
- mark[3] is equal to 17
- mark[4] is equal to 9

Change Value of Array elements

- int mark[5] = {19, 10, 8, 17, 9};

- // make the value of the third element to -1
- mark[2] = -1;

- // make the value of the fifth element to 0
- mark[4] = 0;

mark[0]=19 mark[1]=10 mark[2]=-1 mark[3]=17 mark[4]=0

Array Input/Output

```
/ Program to take 5 values from the user
and store them in an array
// Print the elements stored in the array
#include <stdio.h>
#include <conio.h>
void main()
  {
 int values[5], i;
 printf("Enter 5 integers: ");
 // taking input and storing it in an array
for( i = 0; i < 5; ++i)
 {
    scanf("%d", &values[i]);
  }
  printf("Displaying integers: ");
  // printing elements of an array
 for(int i = 0; i < 5; ++i)
 {
    printf("%d\n", values[i]);
  }
 getch();
}
```

Array Input/Output

- Here, we have used a for loop to take 5 inputs from the user and store them in an array.
- Then, using another for loop, these elements are displayed on the screen.

PROGRAMS

1. Calculate average of n numbers using array.

Access elements out of bound

- Suppose you declared an array of 10 elements. Let's say,

```
int testArray[10];
```

You can access the array elements from testArray[0] to testArray[9]

- Now let's say if you try to access testArray[12]. The element is not available.
- This may cause unexpected output (undefined behavior). Sometimes you might get an error and some other time your program may run correctly.

- Hence, you should never access elements of an array outside of its bound.

Two-dimensional arrays

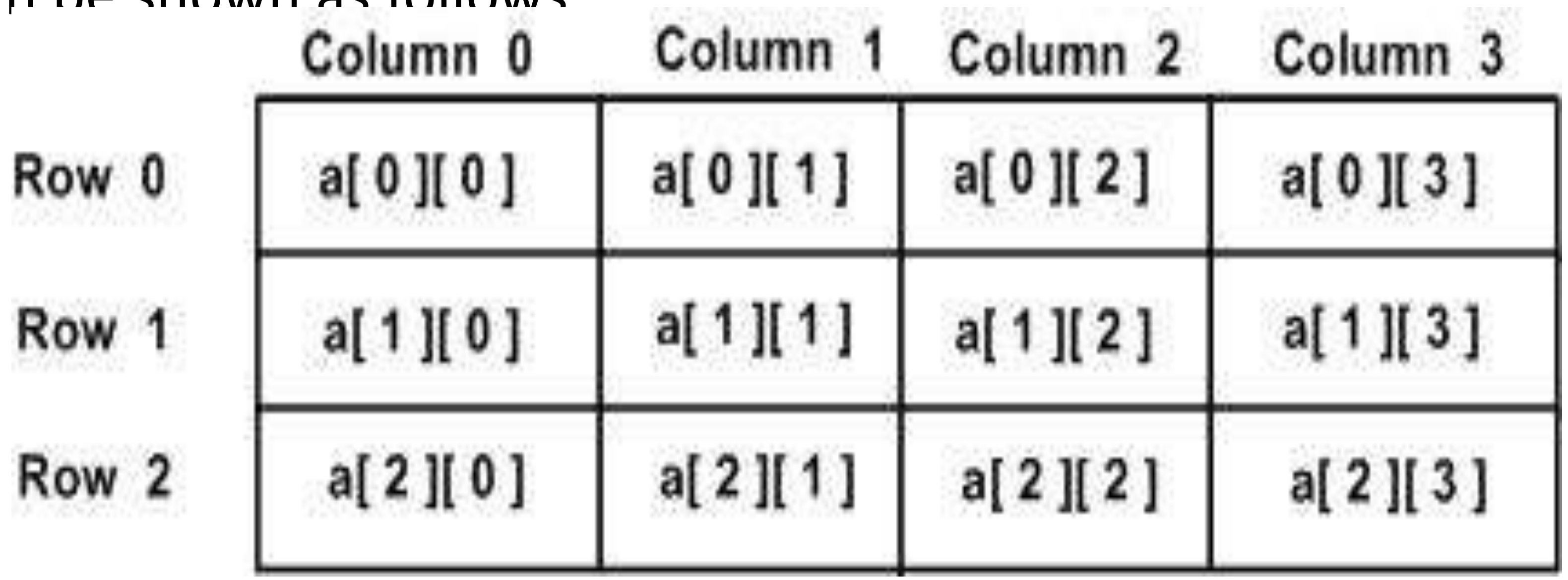
- C programming language allows multidimensional arrays. Here is the general form of a multidimensional array declaration −
- type name[size1][size2]...[sizeN]; For example, the following declaration creates a three dimensional integer array −

```
int threedim [5][10][4];
```

- The simplest form of multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size [x][y], you would write something as follows −

```
type arrayName [ x ][ y ];
```

- Where type can be any valid C data type and arrayName will be a valid C identifier. A two-dimensional array can be considered as a table which will have x number of rows and y number of columns. A two-dimensional array a, which contains three rows and four columns can be shown as follows



	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Thus, every element in the array a is identified by an element name of the form a[i][j], where 'a' is the name of the array, and 'i' and 'j' are the subscripts that uniquely identify each element in 'a'.

Initializing Two-Dimensional Arrays

- Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row has 4 columns.

- int a[3][4] = {

```
 {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
  {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
  {8, 9, 10, 11}   /*  initializers for row indexed by 2 */
  };
```

- The nested braces, which indicate the intended row, are optional. The following initialization is equivalent to the previous example −

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

- **Initialization of a 3d array**

```
int test[2][3][4] = {
 {{3, 4, 2, 3}, {0, -3, 9, 11}, {23, 12, 23, 2}},
{{13, 4, 56, 3}, {5, 9, 3, 5}, {3, 1, 4, 9}}};
```

Two dimensional array Input / Output

```
/ Program to store values in a two
dimensional array and to print the
elements stored in the array/
#include <stdio.h>
#include <conio.h>
void main()
  {
 int values[3] [3], i,j;
 printf("Enter the values\n ");
 // taking input and storing it in the
array
for( i = 0; i < 3; ++i)
for(j=0; j<3; ++j)
    scanf("%d", &values[i][j]);
printf("Displaying integers: ");
 // printing elements of the array
 for( i = 0; i < 3; ++i)
{
for( j = 0; j < 3; ++j)
 printf("%d", values[i][j]);
 printf("\n");
}
 getch();
}
```

Two-dimensional Array Input/Output

```
// C program to store temperature of two cities of a
week and display it.
#include <stdio.h>
const int CITY = 2; const int WEEK = 7;
void  main()
{
  int temperature[CITY][WEEK] ,i, j;
  // Using nested loop to store values in a 2D array
  for ( i = 0; i < CITY; ++i)
  {
    for (int j = 0; j < WEEK; ++j)
    {
      printf("City %d, Day %d:\n ", i + 1, j + 1);
      scanf("%d", &temperature[i][j]);
    }
  }
printf("\nDisplaying values: \n\n");
// Using nested loop to display values of a 2D
array
  for (int i = 0; i < CITY; ++i)
  {
  printf(“City  %d \n”,i+1)
    for (int j = 0; j < WEEK; ++j)
    {
      printf(" Day = %d  temperature= %d\n", i
+ 1, temperature[i][j]);
    }
printf(“\n\n”);
  }
  return 0;
}
```

Sum of Two Arrays

```
#include <stdio.h>
#include <conio.h>
void main()
{
  float a[2][2], b[2][2], result[2][2];
  // Taking input using nested for loop
  printf("Enter elements of 1st matrix\n");
  for (int i = 0; i < 2; ++i)
  for (int j = 0; j < 2; ++j)
    {
      printf("Enter a%d%d: ", i + 1, j + 1);
      scanf("%f", &a[i][j]);
    }
// Taking input using nested for loop
  printf("\nEnter elements of 2nd matrix\n");
   for (int i = 0; i < 2; ++i)
    for (int j = 0; j < 2; ++j)
    {
      printf("Enter b%d%d: ", i + 1, j + 1);
      scanf("%f", &b[i][j]);
     }
// adding corresponding elements of two arrays
  for (int i = 0; i < 2; ++i)
    for (int j = 0; j < 2; ++j)
    {
      result[i][j] = a[i][j] + b[i][j];
    }
/ Displaying the sum
  printf("\nSum Of Matrix:");
  for (int i = 0; i < 2; ++i)
    for (int j = 0; j < 2; ++j)
          printf("%.1f\t", result[i][j]);
        printf("\n");
    }
  getch();
}
```

Programs using arrays

- Linear sorting
- Bubble sort
- Median of n numbers
- Standard Deviation of numbers
- Program to find the largest and second largest number in a set of numbers
- Program to delete an element from an array in a given position
- Program to add an element at a given position in the array.
- Linear search

Programs using arrays

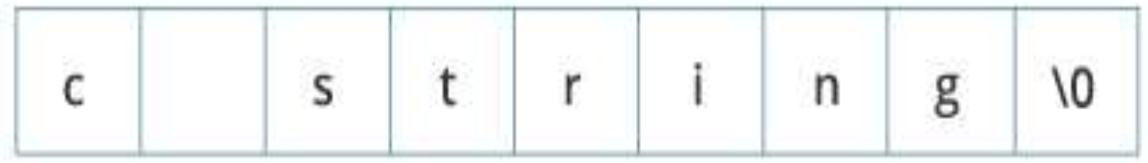
- Binary search
- Merging of two arrays
- Sum of main diagonal and off diagonal elements
- Matrix is symmetric or not.
- Product of 2 matrices
- Print the elements of upper triangle and lower triangle in a squre matrix
- Program to find the determinant of a matrix.

Strings in C

- In C programming, a string is a sequence of characters terminated with a null character \0. For example:

char c[] = "c string";

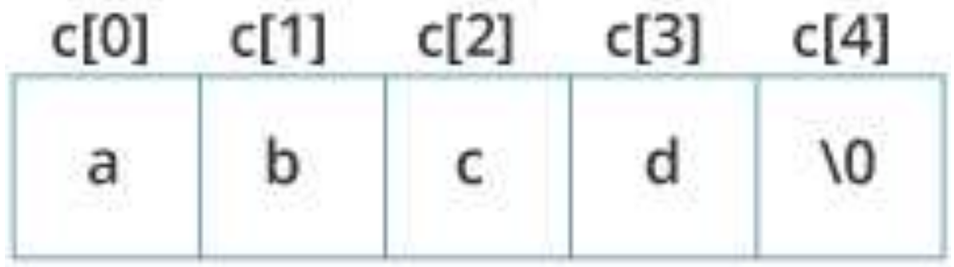
When the compiler encounters a sequence of characters enclosed in the double quotation marks, it appends a null character \0 at the end by default.



How to initialize strings?

You can initialize strings in a number of ways.

- char c[] = "abcd";
- char c[50] = "abcd";
- char c[] = {'a', 'b', 'c', 'd', '\0'};
- char c[5] = {'a', 'b', 'c', 'd', '\0'};



Let's take another example:

```
char c[5] = "abcde";
```

Here, we are trying to assign 6 characters (the last character is '\0') to a char array having 5 characters.

This is incorrect and you should never do this.

Assigning Values to Strings

- Strings do not support the assignment operator once it is declared. For example,

char c[100];

- c = "C programming"; // Error! array type is not assignable.
- Note: Use the strcpy() function to copy the string instead.

Commonly Used String Functions

- **Commonly Used String Functions**
- **strlen()** - calculates the length of a string
- **strcpy()** - copies a string to another
- **strcmp()** - compares two strings
- **strcat()** - concatenates two strings

C strlen()

- The strlen() function calculates the length of a given string.
- The strlen() function takes a string as an argument and returns its length. The returned value is of the unsigned integer type.
- It is defined in the <string.h> header file.
- Note that the strlen() function doesn't count the null character \0 while calculating the length.

Example: C strlen() function

```
#include <stdio.h>
#include<conio.h>
#include <string.h>
 void  main()
{
    char a[20]="Program";
    char a[20] = {'P','r','o','g','r','a','m','\0'};
     char b[10]=“my home”;
printf("Length of string a = %d
\n",strlen(a));
    printf("Length of string b = %d
\n",strlen(b));
strcpy(a,b);
    getch();
}
```

C strcpy()

- strcpy() function in C programming is used to copy strings
- The strcpy() function copies the string pointed by source (including the null character) to the destination.
- The strcpy() function also returns the copied string.

The strcpy() function copies the string source (including the null character) to the string destination.

```
strcpy( destination, source);
```

strcmp() function in C

- The strcmp() function takes two strings and returns an integer.
- The strcmp() compares two strings character by character.
- If the first character of two strings is equal, the next character of two strings are compared. This continues until the corresponding characters of two strings are different or a null character '\0' is reached.
- It is defined in the string.h header file.

strcmp() function in C

Return Value from strcmp()

Return Value	Remarks
0	if both strings are identical (equal)
negative	if the ASCII value of the first unmatched character is less than the second.
positive integer	if the ASCII value of the first unmatched character is greater than the second.

sat   sit   s-s=0 a-i= -ve value

Example: C strcmp() function

```
#include <stdio.h>
#include<conio.h>
#include <string.h>
void main()
{
   char str1[] = "abcd", str2[] = "abCd", str3[] = "abcd";
   int result;
   // comparing strings str1 and str2
   result = strcmp(str1, str2);
   printf("strcmp(str1, str2) = %d\n", result);
   // comparing strings str1 and str3
   result = strcmp(str1, str3);
   printf("strcmp(str1, str3) = %d\n", result);
   getch();
}
```

C strcmp() function

Output

strcmp(str1, str2) = 32>0 positive (str2 is alphabetically above str1)

strcmp(str1, str3) = 0(equal to zero) (str1 is equal to str2)

The first unmatched character between string str1 and str2 is third character. The ASCII value of 'c' is 99 and the ASCII value of 'C' is 67. Hence, when strings str1 and str2 are compared, the return value is 32 (99-67).

When strings str1 and str3 are compared, the result is 0 because both strings are identical.

When strings str1 and str2 are compared and if the result is negative, str2 is alphabetically below str1.

C strcat()

- In C programming, the strcat() function concatenates (joins) two strings.
- The function definition of strcat() is:

  strcat(destination , source)
- As you can see, the strcat() function takes two arguments:

  destination - destination string

  source - source string
- The strcat() function concatenates the destination string and the source string, and the result is stored in the destination string.

Example: C strcat() function

```
#include <stdio.h>
#include<conio.h>
#include <string.h>
void main()
 {
   char str1[100] = "This is ", str2[] = "programiz.com";
   // concatenates str1 and str2
   // the resultant string is stored in str1.
   strcat(str1, str2);
   puts(str1);
   puts(str2);
   getch();
}
```

Output

```
This is programiz.com

programiz.com
```

Note: When we use strcat(), the size of the destination string should be large enough to store the resultant string. If not, we will get the segmentation fault error.

Program to copy one string to another without using strcpy()

```
#include <stdio.h>
#include<conio.h>
void main()
{
   char s1[100], s2[100], i;
   printf("Enter string s1: ");
   gets(s1);

   for (i = 0; s1[i] != '\0'; ++i)
      s2[i] = s1[i];

   s2[i] = '\0';
   printf("String s2: %s", s2);
  getch();
}
```

Program to find the length of a string without using strlen()

```
#include <stdio.h>
 void main()
 {
   char s[] = "Programming is fun";
   int i;
   for (i = 0; s[i] != '\0'; ++i);
   printf("Length of the string: %d", i);
   getch();
}
```

C Program to Concatenate Two Strings without using strcat()

```
#include <stdio.h>
#include<conio.h>
void main()
 {
  char s1[100] = "programming ", s2[] =
"is awesome";
  int length, j;
  // store length of s1 in the length
variable
  length = 0;
  while (s1[length] != '\0') {
    ++length;
  }
 // concatenate s2 to s1
  for (j = 0; s2[j] != '\0'; ++j, ++length) {
    s1[length] = s2[j];
  }
// terminating the s1 string
  s1[length] = '\0';

  printf("After concatenation: ");
  puts(s1);

  return 0;
}
```

C program to compare two strings without strcmp() function

```
#include <stdio.h>
#include <string.h>
#include<conio.h>
void main()
{
  char str1[100], str2[100];
  int  i, flag;
 printf("\n Please Enter the First String : ");
 gets(Str1);
printf("\n Please enter the Second String :  ");
 gets(Str2);
for(i=0;(str1[i]!=‘\0’)||(str2[i]!=‘\0’));++i)
{
If(str1[i]==str2[i])
flag=1;
else If(str1[i]>str2[i])
        {
         flag=0;
         printf(“str1 is alphabetically below str2”);
         }
         else
         {
          printf(“str1 is alphabetically above str2”);
         }
}
If (flag==1)
printf(“strings are equal”);
getch();
}
```

Questions

- Write a program to read lines of text and print it.
- Write a program to convert lines of text in uppercase if the characters are in lowercase.
- Program to find the number of lines, digits , alphabets and number of characters in a text.
- Program to find the number of words in a text.
- Program to check if a string is palindrome or not.
- Program to sort strings in alphabetical order.
- Program to sort list of names without considering case sensitivity
- Program to count occurances of all words.

Questions

- Interchange a word with another word in a text.
- Find the largest word in a given text.
- Without using library function change a line of text to uppercase.
- Program to delete a particular word from a string.
- Enter two strings. Check whether the second string is the substring of the first.

FUNCTIONS

- A function is a group of statements that together perform a task.
- Every C program has at least one function, which is main().
- We can divide up our code into separate functions (user defined functions).
- How you divide up your code among different functions is up to you, but logically the division is such that each function performs a specific task.
- A function can also be referred as a method or a sub-routine or a procedure, etc.
- The C standard library provides numerous built-in functions that your program can call. For example, **strcat()**

- A function consists of three parts:
  - Function definition: provides the actual body of the function.
  - Function declaration: Tells the compiler about a function's name, return type, and parameters
  - Function call: to use a function, a function has to be called.

Function definition

- The general form of a function definition in C programming language is as follows

```
 return_type  function_name( parameter list )
{
 body of the function
}
```

- A function definition in C programming consists of a function header and a function body.
- All the parts of a function header are:
  - Return type
  - Function name
  - Parameter list

Function Header and Function body

- Return Type − A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.
- Function Name − This is the actual name of the function.
- Parameters − A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

- Function Body – The function body contains a collection of statements that define what the function does.

Example: Function called **max()**. This function takes two parameters num1 and num2 and returns the maximum value between the two.

```
int max(int num1, int num2)
 {
  /* local variable declaration */
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

Function Declarations

- A function declaration tells the compiler about a function name and how to call the function.
- A function declaration has the following parts

```
return_type function_name( parameter list );
```

- For the above defined function max(), the function declaration is as follows

```
int max(int num1, int num2);
```

- Parameter names are not important in function declaration only their type is required, so the following is also a valid declaration

```
int max(int, int);
```

Calling a Function

- To use a function, you will have to call that function to perform the defined task.

- When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.

- To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value.

Example:

```
#include <stdio.h>
#include<conio.h>
/* function declaration */
  int max(int num1, int num2);
  void main ()
  {
  /* local variable definition */
  int a = 100;
  int b = 200;
  int ret;
   /* calling a function to get max value */
   ret = max(a, b);
 printf( "Max value is : %d\n", ret );
   getch();
}

int max(int num1, int num2)
 {

   /* local variable declaration */
   int result;

   if (num1 > num2)
     result = num1;
   else
     result = num2;

   return result;
}
```

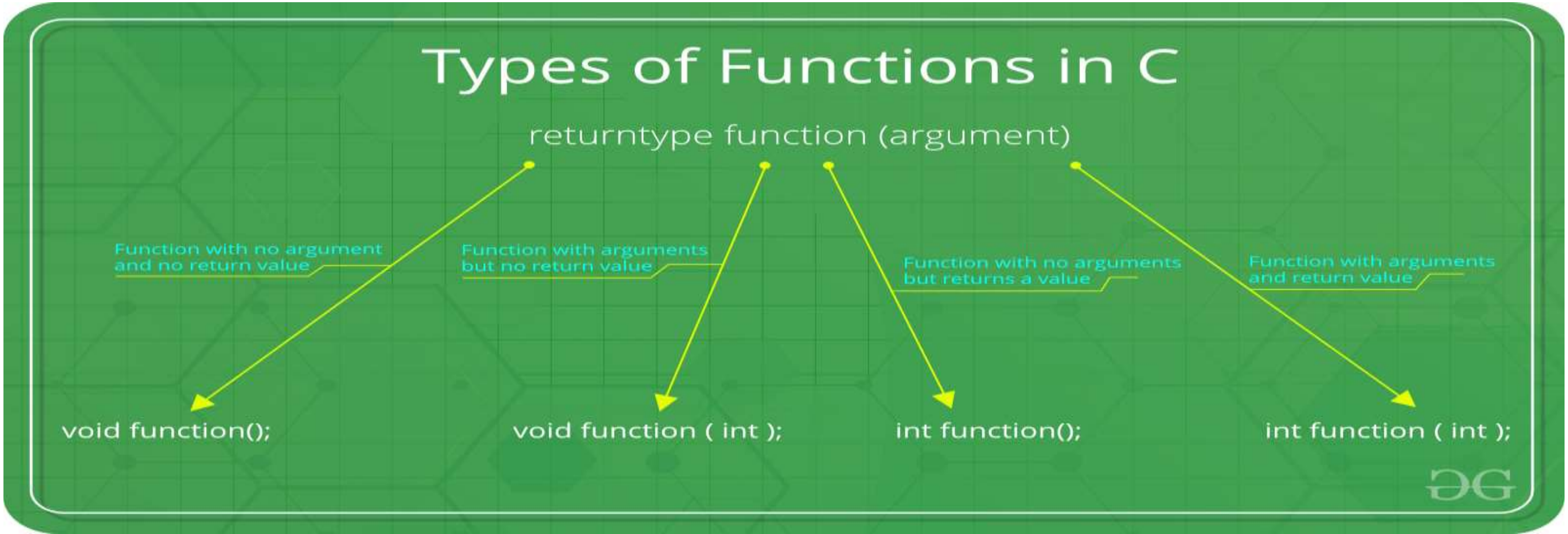
Local declarations and global declarations

- Local declarations:
  - these declarations are made inside the functions.
  - They are known only to functions inside which it is made.

- Global declarations:
  - These declarations are made before main() function
  - They are known to main() function and all the other user defined functions used in the program.

C function argument and return values

- A function in C can be called either with arguments or without arguments.
- These function may or may not return values to the calling functions.
- **There are following categories:**


Types of Functions in C
returntype function (argument)
Function with no argument and no return value
Function with arguments but no return value
Function with no arguments but returns a value
Function with arguments and return value
void function();
void function (int);
int function();
int function (int);

Case 1: Function with no argument and no return value

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.

```
#include <stdio.h>
#include<conio.h>
void mult (void)
int c,d;
void main()
{
printf(enter the no.\n”);
scanf(”%d%d”,&c,&d);
mult();
getch();
}
void mult(void)
{
int e;
e= c*d;
printf (“product is %d”, e);
}
```

Case 2: Function with arguments but no return value

When a function has arguments, it receives the data from the calling function but it returns no values.

```
#include <stdio.h>
#include<conio.h>
void mult (int a, int b)
void main()
{
int c,d;
printf(enter the nos.\n”);
scanf(”%d%d”,&c,&d);
mult(c,d);
getch();
}
void mult(int a, int b)
{
int e;
e= c*d;
printf (“product is %d”, e);
}
```

Case 3: Function with no arguments but return 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

```
#include <stdio.h>
#include<conio.h>
int mult (void)
int c,d;
void main()
{
int z;
printf(enter the nos.\n”);
scanf(”%d%d”,&c,&d);
z=mult(c,d);
printf (“product is %d”, z);
getch();
}
int mult(void)
{
int e;
e= c*d;
return(e);
}
```

Case 4: Function with arguments and return values

These are functions that are designed to take data from the calling functions and also returns a value to the calling function

```
#include <stdio.h>
#include<conio.h>
int mult (int a, int b);
void main()
{
int c,d, z;
printf(enter the nos.\n”);
scanf(”%d%d”,&c,&d);
z=mult(c,d);
printf (“product is %d”, z);
getch();
}
int mult(int a, int b)
{
int e;
e= a*b;
return(e);
}
```

Questions

- Program to find NC_r
- Program to calculate $1 + \frac{x}{1!} + \frac{x^2}{2!} + \cdots$

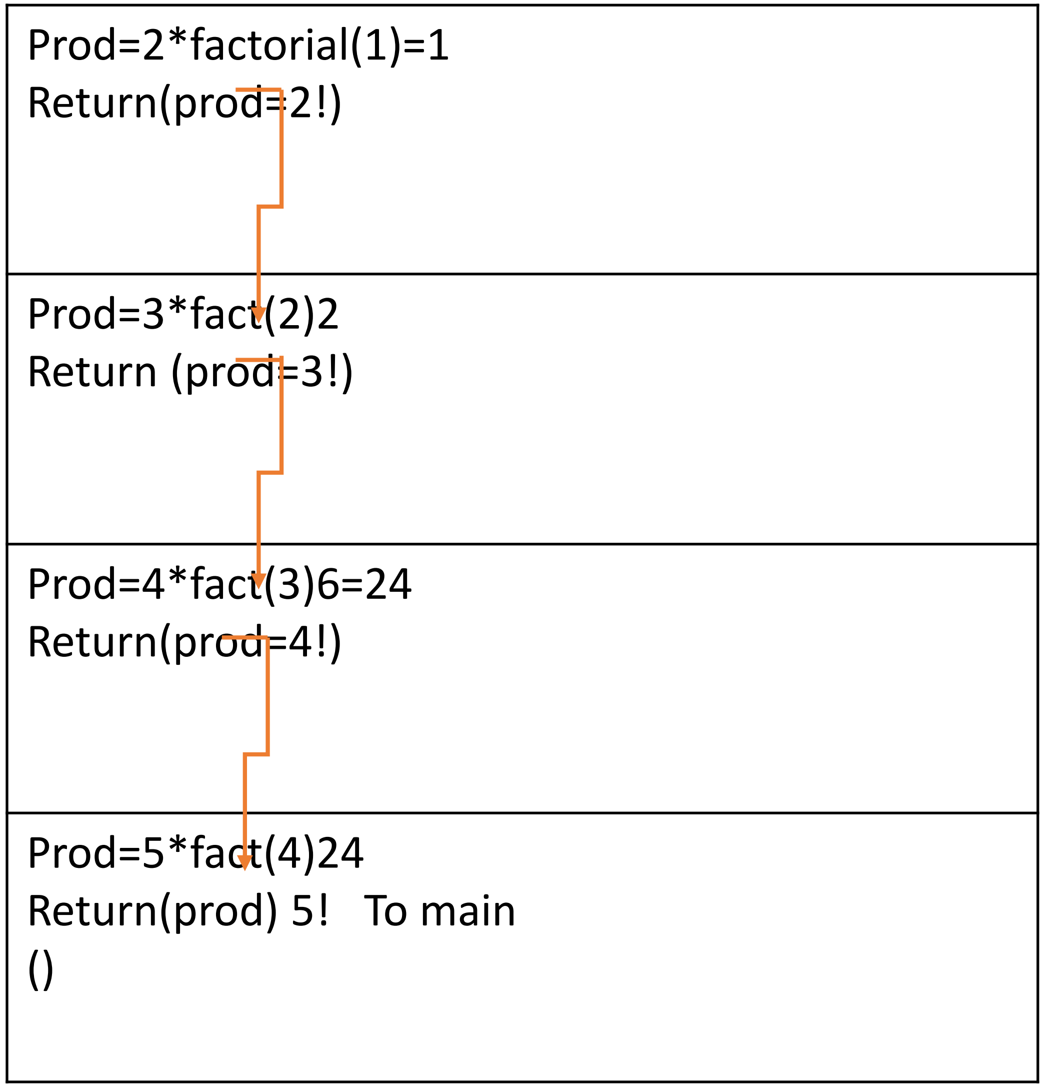
Recursion in C

- A function that calls itself in C is called recursion.
- Recursive functions are very useful to solve many mathematical problems, such as calculating the factorial of a number, generating Fibonacci series, etc.
- Through **Recursion** one can solve problems in easy way while its iterative solution is very big .
- Recursive functions make use of stacks.

Number Factorial using Recursion

```
#include <stdio.h>
#include<conio.h>
long int factorial(int i);
void main() {
 int i;
printf(“enter a value for i\n”); i=5
scanf(“%d”,&i);
printf("Factorial of %d is %d\n", i, factorial(i));120
 getch();
}
long int factorial(int i) 1
{
int prod;
 if(i <= 1)                              prod=5xfactorial(4)
     return 1;                            prod=4xfactorial(3)     =3!x4
else                                     prod=3xfactorial(2)      =2!x3
   prod= ( i * factorial(i – 1));      =1!x2
return(prod);
```

STACK



Prod=2*factorial(1)=1
Return(prod=2!)

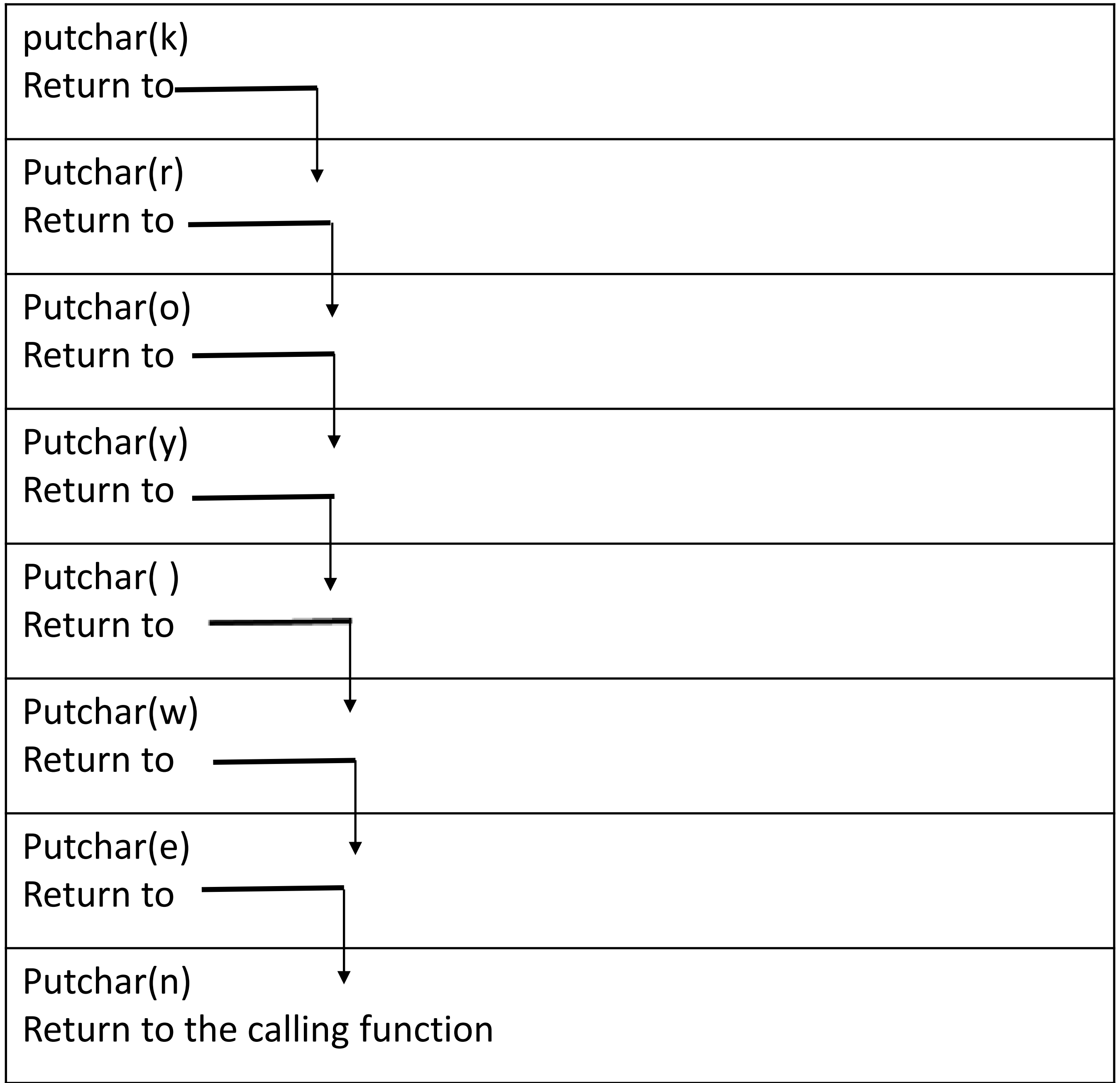
Prod=3*fact(2)2
Return (prod=3!)

Prod=4*fact(3)6=24
Return(prod=4!)

Prod=5*fact(4)24
Return(prod) 5!   To main
()

Program to reverse a line of text

```
#include<stdio.h>
#include<conio.h>
void reverse(void);
void main()                          enter line
{                              new york
printf("ënter line\n");   kroy wen
reverse();
getch();
}
void reverse(void)
{
char c;
If((c=getchar())!='\n')
reverse();
putchar(c);
return;
}
```



Questions

- Write a program for binary search using recursion
- Print the first n natural nos. in reverse order
- Print Fibonacci numbers using recursion
- Find the sum of any n numbers using recursion
- Write a program in C to find the sum of digits of a number using recursion.
- Write a program in C to count the digits of a given number using recursion
- Write a program in C to convert a decimal number to binary using recursion.

Passing Arrays to Functions

Single Dimensional Arrays

- The array should be followed by square bracket.
- No need to declare the size of the array in function declaration and function definition

```
int search(int a[]);
```

- Mention only the name of the array in function call.

```
val=search(a);
```

Two dimensional Arrays

- We must mention the second dimension in function declaration and function definition.

```
void read(int d[][5],int r, int s);
```

- Mention only the name of the array in function call.

QUESTIONS

- Program to find S.D. of a set of numbers with 2 functions, one to find mean and other to find S.D.
- Program to sort array using function
- Program to find product of 2 matrices
- Program to find $1 + \frac{x}{1!} + \frac{x^2}{2!} + \cdots$ using two functions –one to find power and another for factorial.
- Sum of upper triangle elements
- Sum of off diagonal and main diagonal elements
- Matrix symmetric or not using functions
- To arrange rows of array in ascending order
- Determinant of 3x3 matrix

POINTERS IN C

- A **pointer** is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address. The general form of a pointer variable declaration is

```
  type *var-name;
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```

How to use pointers?

- There are a few important operations, which we will do with the help of pointers very frequently.

  (a) We define a pointer variable,

  (b) assign the address of a variable to a pointer and

  (c) finally access the value at the address available in the pointer variable.

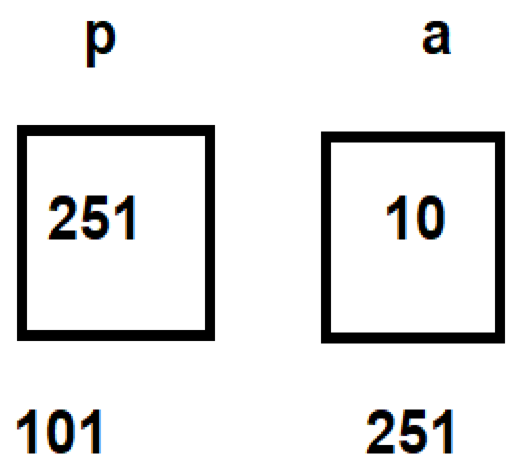
  This is done by using unary operator ***** that returns the value of the variable located at the address specified by its operand.

The following example makes use of these operations

```
#include <stdio.h>

void main ()
{
   int  var = 20;   /* actual variable declaration */
   int  *ip;        /* pointer variable declaration */

   ip = &var;  /* store address of var in pointer variable*/
   printf("Address of var variable: %x\n", &var  );
  /* address stored in pointer variable */
   printf("Address stored in ip variable: %x\n", ip );
   /* access the value using the pointer */
   printf("Value of *ip variable: %d\n", *ip );
   }
```



p=251
*p=10

```
void main()
{
int *p,a=10;
clrscr();
p=&a; (p=251)
print("%d",*p); (*p=10)
getch();
}
```

here at the declaration *p means --->p is a pointer and it can store the address of a variable whose data type is integer

here in the execution part *p means--->value at the address contained in p

here, *p

NULL Pointers

- It is always a good practice to assign a NULL value to a pointer variable in case you do not have an exact address to be assigned.
- This is done at the time of variable declaration. A pointer that is assigned NULL is called a null pointer. But by convention, if a pointer contains the null (zero) value, it is assumed to point to nothing.
- The NULL pointer is a constant with a value of zero defined in several standard libraries. Consider the following program −

```
#include <stdio.h>

void main ()
 {

   int  *ptr = NULL;

   printf("The value of ptr is : %d\n", ptr  );

   getch();
}
```

When the above code is compiled and executed, it produces the following result −The value of ptr is 0

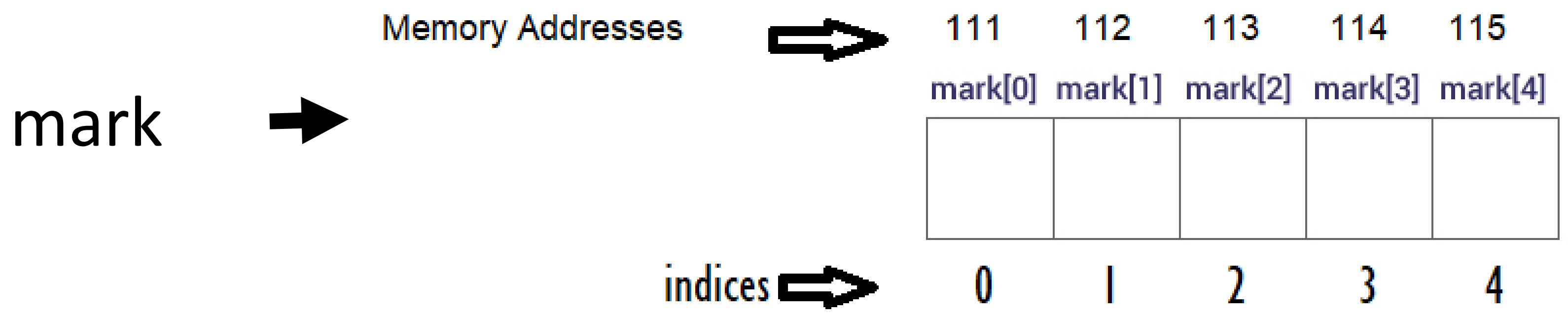
```
if(ptr)     /* succeeds if p is not null */
if(!ptr)    /* succeeds if p is null */
```

POINTERS AND ARRAYS

- The array name stores the address of the first element of the array.

```
 void main()
{
Int a[10];
 here,  the array name a contains the address of a[0] that is,a=&a[0]
a+1=&a[1]
a+2=&a[2]
a+3=&a[3]   and so on
```

POINTERS AND ARRAYS



Here, array mark contains the address of mark[0] which is 111

POINTERS AND ARRAYS

mark=&mark[0]	*(mark)=mark[0]
mark+1= &mark[1]	*(mark+1)=*(&mark[1])=mark[1]
mark+2= &mark[2]	*(mark+2)=*(&mark[2])=mark[2]
mark+3= &mark[3]	*(mark+3)=*(&mark[3])=mark[3]
mark+4= &mark[4]	*(mark+4)=*(&mark[4])=mark[4]

POINTERS AND ARRAYS

int a[5][5]

Name array a represents the address of the first row.

a+i= represents the address of the ith row

*(a+i)= represents the address of the first element of the ith row

*(a+i)+j = represents the address of the jth element of the ith row

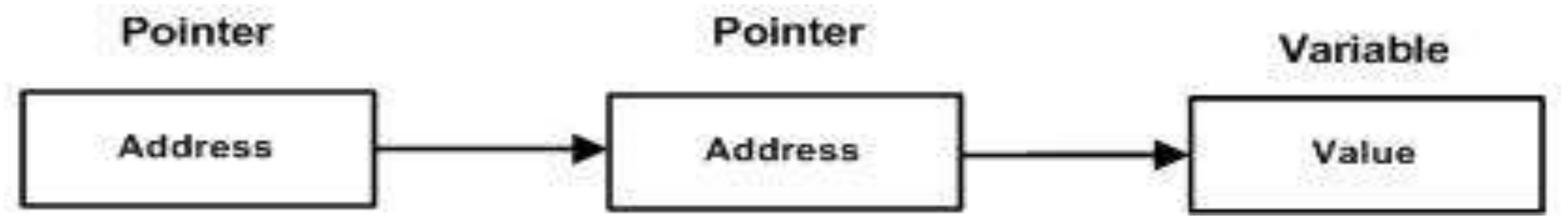
((a+i)+j)= represents the address of the jth element of the ith row

a[3][4]=*(*(a+3)+4)   &a[3][4]=*(a+3)+4

Pointer to Pointers

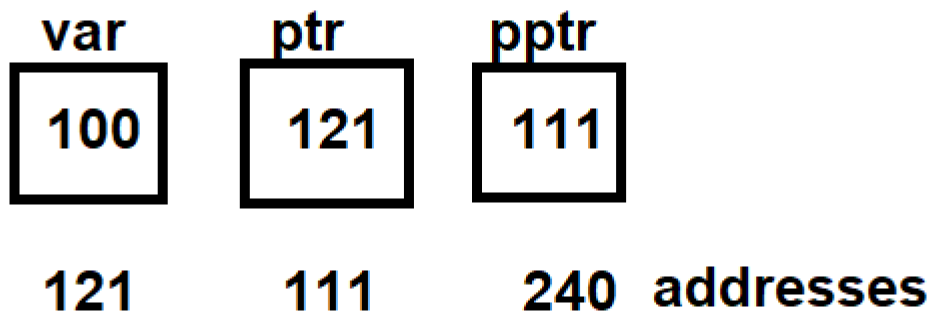
- A pointer to a pointer is a chain of pointers. Normally, a pointer contains the address of a variable.
- When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value as shown below.
- A variable that is a pointer to a pointer must be declared as such. This is done by placing an additional asterisk in front of its name. For example, the following declaration declares a pointer to a pointer of type int −

```
int **var;
```



Example

```
include <stdio.h>
void main ()
 {   int  var;
   int  *ptr;
   int  **pptr;
clrscr();
 var = 3000;
ptr = &var;
pptr = &ptr;
  printf("Value of var = %d\n", var );
   printf("Value available at *ptr = %d\n", *ptr );
   printf("Value available at **pptr = %d\n", **pptr);
}
```



ptr=&var =121
pptr=&ptr=111
***ptr= 100**
***pptr= value at address contained in pptr. 121**
****pptr= value at address contained pptr value at 121 =100**

Arithmetic Operations with Pointers

- Increment ,decrement prefix and postfix operations are performed with pointers.i.e. ++ptr,ptr++,--ptr,ptr--
- Addition, multiplication, modulus and division of pointers not possible. That is, arithmetic operations are impossible with addresses. Only the value stored at addresses can be used for addition, multiplication and division.

Structures

- These are data types that can store elements of different data types.
- Structures are derived data types.
- Structures can be declared as follows

```
Struct struct_type
{
type variable1;
type variable2;
….
……
};
```

Here, struct_type is known as tag. variable 1, variable 2 etc. are the members of the structure template. After defining structure template, we can create variables as given below.

```
struct struct_type v1,v2,v3;
```

Structures

- Here, v1,v2 and v3 are variables or objects of structure struct_type.

```
Struct book1
{
char book[30];
int pages;
float price;
};
struct book1 bk1,bk2;
```

- Here, a structure of book1 is created. It consists of three members. Book[30] of char type, pages of int type and price of float type.
- bk1 is a variable of type book1 which is a structure.

Structures

bk1 can be initialized with values as follows:

```
struct book1 bk1={“the lost ring”,500,285};
```

The period (.) sign is used to access structure members.

```
bk1.book=“the lost ring”;
bk1.pages=500;
bk1.price=385;
```

```
bk1.book=“the lost ring”;
bk1.pages=500;
bk1.price=385;
```

Also, two variables of the same structure type can be copied in the same way as ordinary variables.

```
bk1=bk2;
```

ARRAY OF STRUCTURES

- In array of structures, every element of the array is a structure.
- Array of structures can be declared as follows:

```
Struct stud
{ int roll;
 int mark;
 char name[10];
};
struct stud s[70];
```

S[70] is an array containing 70 elements where each element is a structure of type stud.

Pointer to structures

- We can also define a pointer to a structure.
- Such pointers are called structure pointers.

```
Struct book
{
char name[25];
char author[25];
int pages;
int price;
};
struct book  bk1,*ptr;
```

Here, ptr is a pointer to a structure of type book.

This means, ptr contains the address of a variable which is a structure of type book.

Pointer to structures

ptr= &bk1;

In such cases,

bk1.name=ptr->name

bk1.author=ptr->author

bk1.pages=ptr->pages

bk1.price=ptr->price

Passing structures to functions

Create a structure of customers in a bank with name, account number and balance.

Call by value method

- Call by value method - copies the value of an argument(actual parameter) into the formal parameter of that function.
- Therefore, changes made to the formal parameters in the function do not affect the actual parameter of the main function.
- In this call by value method, values of actual parameters are copied to function's formal parameters, and the formal parameters are stored in different memory locations. So any changes made inside functions are not reflected in actual parameters of the caller.

Call by reference method

- Call by reference method copies the address of an argument into the formal parameter. In this method, the address is used to access the actual argument used in the function call. It means that changes made in the parameter alter the passing argument.

- In this method, the memory allocation is the same as the actual parameters. All the operation in the function are performed on the value stored at the address of the actual parameter, and the modified value will be stored at the same address.

CALL BY VALUE Vs CALL BY REFERENCE

Parameters	**Call by value**	**Call by reference**
Definition	While calling a function, when you pass values by copying variables, it is known as "Call By Values."	While calling a function, in programming language instead of copying the values of variables, the address of the variables is used it is known as "Call By References.
Arguments	In this method, a copy of the variable is passed.	In this method, a variable itself is passed.
Effect	Changes made in a copy of variable never modify the value of variable outside the function.	Change in the variable also affects the value of the variable outside the function.
Alteration of value	Does not allow you to make any changes in the actual variables.	Allows you to make changes in the values of variables by using function calls.

CALL BY VALUE Vs CALL BY REFERENCE

Alteration of value	Does not allow you to make any changes in the actual variables.	Allows you to make changes in the values of variables by using function calls.
Passing of variable	Values of variables are passed using a straightforward method.	Pointer variables are required to store the address of variables.
Value modification	Original value not modified.	The original value is modified.
Memory Location	Actual and formal arguments will be created in different memory location	Actual and formal arguments will be created in the same memory location
Safety	Actual arguments remain safe as they cannot be modified accidentally.	Actual arguments are not Safe. They can be accidentally modified, so you need to handle arguments operations carefully.
Default	Default in many programming languages like C++.PHP. Visual Basic NET, and C#.	It is supported by most programming languages like JAVA, but not as default.

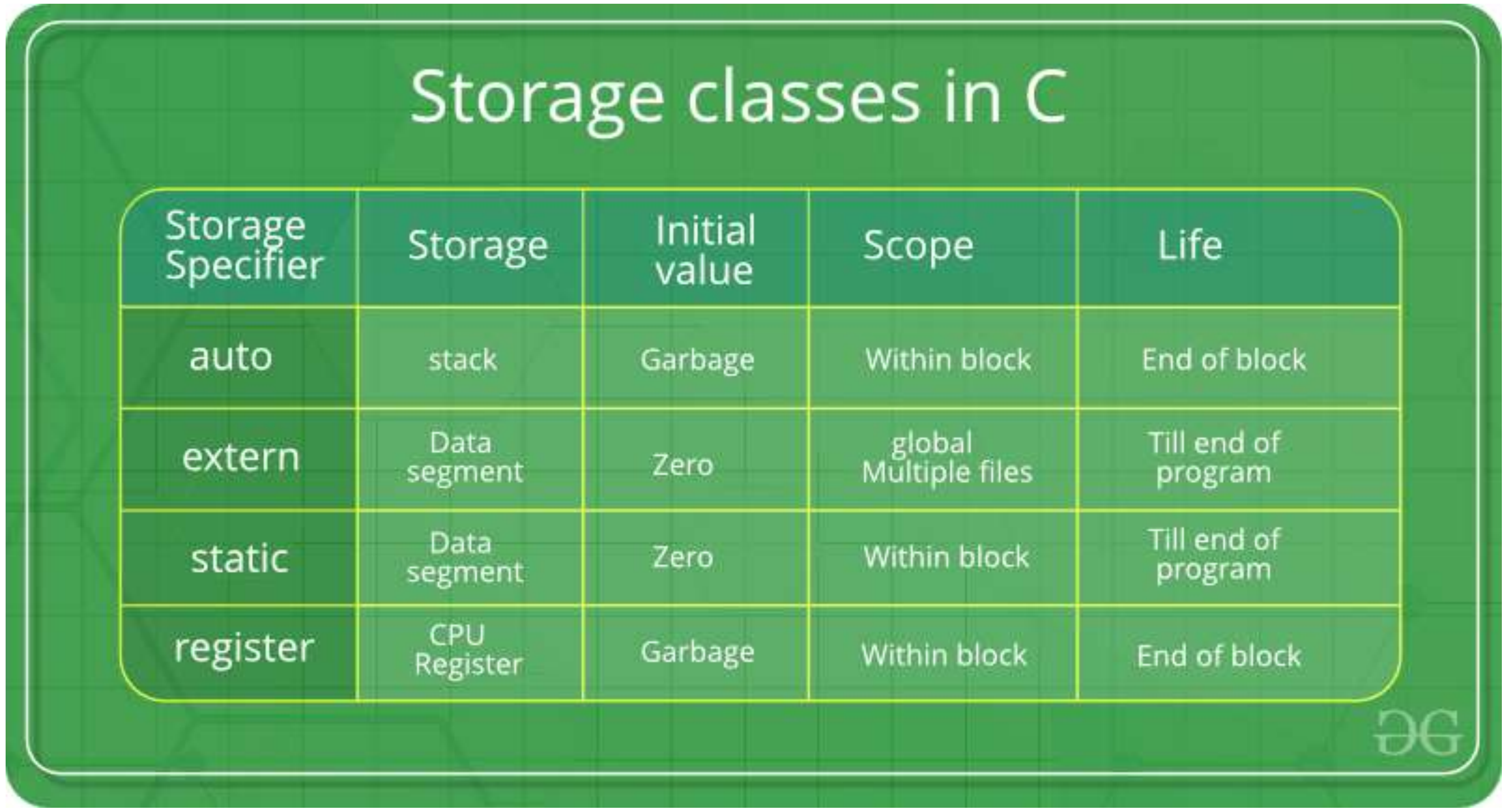
Passing pointers using functions(call by reference)

```
#include<stdio.h>
#include<conio.h>
void swap(int*c,int*d)
void main()
{
int a,b;
clrscr();
printf(“enter value for a\n”);
scanf(“%d”,&a);
printf(“ënter value for b\n”);
scanf(“%d”,&b);
printf(“a=%d”,a);
printf(“b=%d”,b);
getch();
}
void swap(int*c,int*d)
{
int t;
t=*c;
*c=*d;
*d=t:
return;
}
```

STORAGE CLASSES IN C

- Storage Classes are used to describe the features of a variables.
- These features basically include the scope, visibility and life-time which help us to trace the existence of a particular variable during the runtime of a program.
- There are four types of storage classes in C
    - Automatic storage class
    - Extern storage class
    - Static storage class
    - Register storage class

Storage classes in C



Storage classes in C

Storage Specifier	Storage	Initial value	Scope	Life
auto	stack	Garbage	Within block	End of block
extern	Data segment	Zero	global Multiple files	Till end of program
static	Data segment	Zero	Within block	Till end of program
register	CPU Register	Garbage	Within block	End of block

- **auto**: This is the default storage class for all the variables declared inside a function or a block.
- Hence, the keyword auto is rarely used while writing programs in C language.
- Auto variables can be only accessed within the block/function they have been declared and not outside them (which defines their scope
- They are assigned a garbage value by default whenever they are declared.

Extern storage class

- Extern storage class simply tells us that the variable is defined elsewhere and not within the same block where it is used.
- They are also known as global variables.
- Global variables can be accessed by any function in the program.
- External variables are declared outside a function.
- Unlike local variables, global variables are initialized to zero by default.
- Another aspect of global variable is that it is available only from the point of declaration to the end of the program.

Global Variables

```
#include<stdio.h>
#include<conio.h>
int fun1(void);
int fun2(void);
int x;
void main()
{
x=10; /*global*/
printf(“x=%d”,x);
printf(“x=%d”, fun1());
printf(“x=%d”, fun2());
}
```

```
int fun2(void)
{
x=x+1; /*global*/
return(x);
}
int fun1(void)
{
x=x*11; /*global
return(x);
}
```

Global Variables

```
#include<stdio.h>
#include<conio.h>
int fun1(void);
int fun2(void);
int x;
void main()
{
x=10; /*global*/
printf(“x=%d”,x);
printf(“x=%d”, fun1());
printf(“x=%d”, fun2());
}
```

```
int fun2(void)
{
x=x+1; /*global*/
return(x);
}
int fun1(void)
{
x=x*11; /*global
return(x);
}
```

Global Variables

```
#include<stdio.h>
#include<conio.h>
int fun1(void);
int fun2(void);
void main()
{
printf(“x=%d”,x);
printf(“x=%d”, fun1());
printf(“x=%d”, fun2());
}
```

```
int x;
int fun2(void)
{
x=x+1; /*global*/
return(x);
}
int fun1(void)
{
x=x*11; /*global
return(x);
}
```

External Declaration

- In the program above the main() function cannot access the variable x as it is declared after the main() function.
- This problem can be solved by declaring the variable with the storage class extern.

External Declaration

```
#include<stdio.h>
#include<conio.h>
int fun1(void);
int fun2(void);
void main()
{
External int x;
printf("x=%d",x);
printf("x=%d", fun1());
printf("x=%d", fun2());
}
```

```
int x;
int fun2(void)
{
x=x+1; /*global*/
return(x);
}
int fun1(void)
{
x=x*11; /*global
return(x);
}
```

The register Storage Class

- The register storage class is used to define local variables that should be stored in a register instead of RAM.
- This means that the variable has a maximum size equal to the register size (usually one word)

```
register int  miles;
```

- The register should only be used for variables that require quick access such as counters.
- It should also be noted that defining 'register' does not mean that the variable will be stored in a register. It means that it MIGHT be stored in a register depending on hardware and implementation restrictions.

The static Storage Class

- The **static** storage class instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying it each time it comes into and goes out of scope.
- Therefore, making local variables static allows them to maintain their values between function calls.

Fibonacci series using static function

```
#include<stdio.h>
#include<conio.h>
void fib(int a, int b,int n)
{
int f1=0,f2=2,n,’;
clrscr();
printf(ënter the limit\n”);
printf(“%3d%3d”,f1,f2);
fib(f1,f2,n);
}
```

```
void fib(int a,int b,int c)
{
int sum;
static int i=2;
sum=a+b;
a=b;
b=sum;
i=i+1;
printf(“%d”,sum);
If(i<c)
fib(a,b,c);
return;
}
```