### 21.a.i)

Pseudo code:

FUNCTION factorial(INT n)

DECLARE INT res

SET res = 1

FOR i FROM 1 TO n STEP 1

res=res\*i

END FOR

BEGIN Main

DECLARE INT num,result

INPUT num

SET res = factorial(num)

DISPLAY res

END Main

### 21.a.ii)

| Pre-increment | Post-increment |
| --- | --- |
| 1. The value is incremented by 1 right away | 1. The value is incremented by 1 when it encounters the same variable again |
| 2. If variable is : i  Pre-increment :- ++i | 2. If variable is : i  Post-increment :- i++ |
| 3. Eg:  int i=0;  printf(“%d”,++i);  Output:  1 | 3. Eg:  int i=0;  printf(“%d”,i++);  printf(“\n%d”,i);  Output:  0  1 |

### 

#### 

### 21. b)

Logical Operators:

Used to perform logical operations, typically on boolean values (expressions that evaluate to true or false). They are mostly used in control flow statements like if, while etc.

We have 3 types of logical opertors in C:

1. Logical AND : &&
2. Logical OR : ||
3. Logical NOT : !

**&& (logical AND)**: Returns true if both operands are non-zero (i.e., true), otherwise returns false.

**|| (logical OR)**: Returns true if at least one operand is non-zero (i.e., true).

**! (logical NOT):** Inverts the boolean value (i.e., turns true to false and false to true)

Example:

#include<stdio.h>

int main() {

int a=5,b=10,c=50;

if(a<b && a\*b==c) {

printf("%d \* %d equal to %d",a,b,c);

}

else {

printf("%d \* %d not equal to %d",a,b,c);

}

}

Here,

a=5

b=10

c=50

In the IF statement we are checking if both (a<b) and (a\*b==c) conditions are true.

If the condition is true, then the print statement : 5 \* 10 is equal to 50 will be executed

Else condition is false, then the print statement : 5 \* 10 is not equal to 50 will be executed

Similarly,

For OR operator || ,

It will check if either conditions are true

Example:

if(a>b || a\*b==c) : since 2nd condition is true , the if condition will be true

For Not Operator ! ,

It will reverse/invert the boolean value (1 or 0) returned by the condition

if(!(a>b) : returns true since a>b returns 0 but the ! operator inverts it to 1

### 22. a.

Program to check if a number is palindrome or not

#include <stdio.h>

int isPalindrome(int num) {

int original = num;

int reversed = 0;

*// Reverse the number*

while (num > 0) {

int digit = num % 10;

reversed = reversed \* 10 + digit; *// Build the reversed number*

num /= 10;

}

*// Check if the reversed number matches the original*

return (original == reversed);

}

int main() {

int num;

printf("Enter a number: ");

scanf("%d", &num);

if (isPalindrome(num)) {

printf("%d is a palindrome.\n", num);

} else {

printf("%d is not a palindrome.\n", num);

}

return 0;

}

### 22. b.

#include <stdio.h>

#include <stdlib.h>

int main() {

int rows, cols;

printf("Enter the number of rows: ");

scanf("%d", &rows);

printf("Enter the number of columns: ");

scanf("%d", &cols);

*// Dynamically allocate memory for matrices A, B, and result matrix*

int \*\*A = (int \*\*)malloc(rows \* sizeof(int \*));

int \*\*B = (int \*\*)malloc(rows \* sizeof(int \*));

int \*\*result = (int \*\*)malloc(rows \* sizeof(int \*));

for (int i = 0; i < rows; i++) {

A[i] = (int \*)malloc(cols \* sizeof(int));

B[i] = (int \*)malloc(cols \* sizeof(int));

result[i] = (int \*)malloc(cols \* sizeof(int));

}

*// Input elements for matrix A*

printf("Enter elements of matrix A:\n");

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

printf("Enter A[%d][%d]: ", i + 1, j + 1);

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

}

}

*// Input elements for matrix B*

printf("Enter elements of matrix B:\n");

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

printf("Enter B[%d][%d]: ", i + 1, j + 1);

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

}

}

*// Perform matrix addition: result = A + B*

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

result[i][j] = A[i][j] + B[i][j];

}

}

*// Output the result of matrix addition*

printf("\nResultant matrix (A + B):\n");

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

printf("%d ", result[i][j]);

}

printf("\n");

}

return 0;

}

### 23. a.

Four string functions in C are:

1. atoi()

Converts string to number

Eg:

char st[]=”1925”;

int a=atoi(st);

print(“%d”,a)

Output:

1925

1. strlen()

Returns length of string till null character ‘\0’

Eg:

char st[]=”Hello”;

int n=strlen(st);

print(“Length of string is %d”,n);

Output:

Length of string is 5

1. strcat()

Concatenates two string and result is updated in 1st string

Eg:

char s1[]=”Hello “;

char s2[]=”World”;

strcat(s1,s2);

printf(“%s\n %s”,s1,s2);

Output:

Hello World

World

1. strcmp()

Compares two string and returns integer value

If i) 0 then both string equal

ii)>0 then string1 is greater than string2

iii)<0 then string1 is less than string2

### 23. b.

Passing an array to a function falls under Call by reference.

Call by Value :

Actual Parameters : values of variables sent to function

Formal Paramters : values of variables received by variables in the function

Call by Reference :

Actual Parameters : Address of variables sent to function

Fornal Paramters : Address of variables received by pointer variables in the function

When talking about arrays in C,

They are but pointers which points to different addresses that contains a value

Array Declaration:

#include <stdio.h>

#include <stdlib.h>

int main() {

int A[6];

return 0;

}

Here, we declare Array of size 6

But this same thing is possible with pointers.

#include <stdio.h>

#include <stdlib.h>

int main() {

int \*A;

int size=6;

A=(int\*)malloc(size\*sizeof(int));

int i;

for(i=0;i<size;i++) {

printf("A[%d] : ",i+1);

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

}

return 0;

}

Here, we first declare a pointer and a size of 6

To allocate memory of (size \* 4 bytes) we use the malloc() method in stdlib.h

This way we created a 1D array similar to the normal way we declare array

Now for accessing or putting values into the array,

We use the same [i] where i : index position

Example:

#include <stdio.h>

int sum(int\*,int);

int sum(int\* arr,int n) //Formal Parameters

{

int i;

int s=0;

for(i=0;i<n;i++) s+=arr[i];

return s;

}

int main() {

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

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

int s=sum(a,n); //Actual Parameters

printf("%d",s);

return 0;

}

Here, we declared a sum() function:

int sum (int\*,int)

The 1st parameter is pointer

The 2nd parameter is integer

In the main() function,

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

int n=sizeof(a)/sizeof(a[0]); //calculate size of array in bytes

During function call : int s=sum(a,n);

Here, we are passing

1st Parameter as Array a

2nd Parameter as Integer n

When passing the array itself to sum() function, the pointer variable does not copy the values in the array.

When an array is passed to a function, it **decays** into a pointer to its first element. This means you are passing the address of the first element of the array, not the entire array.

But if only the 1st element’s address is passed to the pointer in sum() function,

Then how does it access the other values in array

a={1,2,3}

arr points to a[0]

When you use the indexing syntax arr[i], it’s equivalent to \*(arr + i). This means "take the pointer arr, move it i elements forward in memory, and dereference it to get the value at that location".

Example:

#include <stdio.h>

void printElement(int\* arr) {

printf("arr[0] = %d\n", arr[0]); *// Access the first element*

printf("arr[1] = %d\n", arr[1]); *// Access the second element*

}

int main() {

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

printElement(a); *// Pass the array to the function*

return 0;

}

Explanation:

1. Passing the Array: When you pass the array a to the function printElement(), it decays into a pointer to the first element of the array (a is equivalent to &a[0]).
2. Accessing Elements: arr[0] is equivalent to \*(arr + 0) which accesses the first element of the array (i.e., 1). arr[1] is equivalent to \*(arr + 1) which accesses the second element of the array (i.e., 2).
3. Pointer Arithmetic: The pointer arr initially points to the first element of the array. When you use arr[1], you are moving the pointer to the second element by adding 1 to it. This is how pointer arithmetic allows you to access different elements in the array.

### 24. a.

Four List functions in Python:

1. append() :

The append() function adds an element to the end of the list.

my\_list = [1, 2, 3]

my\_list.append(4) # Adds 4 to the end of the list

print(my\_list) # Output: [1, 2, 3, 4]

Here, we have a list my\_list = [1,2,3]

Using append() function we are adding 4 at the end of the list my\_list

Now, 4 is located at my\_list[3] position

Hence, append() modifies the original list by adding the specified element to the end. This operation does not return any value (returns None), so it's used to directly alter the list

1. remove() :

The remove() function removes the first occurrence of a specified element from the list.

my\_list = [1, 2, 3, 2, 4]

my\_list.remove(2) # Removes the first occurrence of 2

print(my\_list) # Output: [1, 3, 2, 4]

Here, we have a list my\_list = [1,2,3,2,4]

Using remove() function, we are deleting the 1st occurrence number of element 2

Now, my\_list = [1,3,2,4]

If we do : my\_list.remove(2) again

List becomes : my\_list = [1,3,4]

remove() searches for the first occurrence of the element and removes it from the list. If the element is not found, it raises a ValueError.

1. pop() :

The pop() function removes and returns an element from the list at the specified index. If no index is provided, it removes and returns the last element.

my\_list = [10, 20, 30, 40]

popped\_item = my\_list.pop(2) # Removes the element at index 2

print(popped\_item) # Output: 30

print(my\_list) # Output: [10, 20, 40]

Here, we have a list my\_list = [10,20,30,40]

Since pop() works by mentioning the index at which the element is located,

Therefore to remove 30, we mention 2 (since indexing starts from 0)

Now, my\_list = [10,20,40]

pop() removes and returns the item at the given index.

If no index is provided, it removes and returns the last item in the list. If you try to pop() from an empty list, it raises an IndexError

1. sort() :

The sort() function sorts the elements of the list in ascending order by default. It can also sort in descending order or based on a custom key function.

my\_list = [3, 1, 4, 2]

my\_list.sort() # Sorts in ascending order

print(my\_list) # Output: [1, 2, 3, 4]

Here, we have an unsorted list my\_list = [3,1,4,2]

Using sort() function we sort this list in ascending order

Now, my\_list = [1,2,3,4]

But if we want to sort the list in descending order, we do the following

my\_list = [3, 1, 4, 2]

my\_list.sort(reverse=True) # Sorts in descending order

print(my\_list) # Output: [4, 3, 2, 1]

reverse=True argument will tell sort() function to sort this list in descending order.

Now, my\_list = [4,3,2,1]

sort() sorts the list in place, meaning it modifies the original list and does not return a new list.

reverse=True sorts the list in descending order.

But we can also use key argument :

key allows you to define a custom sorting criterion (e.g., sorting by the length of strings)

Eg 1 : Sorting by len

words = ["banana", "pie", "apple", "cherry"]

words.sort(key=len) # Sort by length of the string

print(words)

Output:

['banana', 'cherry', 'apple', 'pie']

Eg 2 : Sorting by abs

numbers = [-10, 5, 8, -3, 7]

numbers.sort(key=abs) # Sort by absolute value

print(numbers)

Output:

[-3, 5, 7, 8, -10]

### 24. b.

def is\_prime(n):

c=0

for i in range(2,n+1):

if(n%i==0): c+=1

return (c==1)

for i in range(1,101):

if(is\_prime(i)):

print(i,end=' ')

Here, we define a is\_prime(n) function to check if n is prime number or not.

Since we are iterating from 1 to 100,

We check if ‘i’ is prime, and print i’s value if its prime number in the output



### 25. a. i)

import pandas as pd

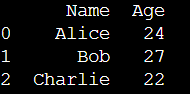
# Example data

data = [['Alice', 24], ['Bob', 27], ['Charlie', 22]]

df = pd.DataFrame(data, columns=['Name', 'Age']) # Create a DataFrame

# Display the DataFrame

print(df)



### 25. a. ii)

| Numpy | Pandas |
| --- | --- |
| Primarily used for numerical operations and handling arrays/matrices. | Primarily used for data manipulation and cleaning data. |
| Works with ndarray, a homogeneous, multi-dimensional array. | Works with DataFrame and Series, which are more flexible, supporting heterogeneous data types (e.g., strings, integers, floats). |
| Faster for numerical operations on large datasets because of its lower-level array operations. | Slightly slower than NumPy for numerical operations due to higher-level abstraction |
| Focuses on mathematical operations like linear algebra, statistics, and array manipulation. | Provides high-level operations for data manipulation, including grouping, merging of data etc. |

### 25. b. i)

# Importing pandas library

import pandas as pd

# Creating 2 pandas Series

ps1 = pd.Series([2.5, 4, 6, 8, 10, 1.75, 40])

ps2 = pd.Series([1.5, 3, 5, 7, 10, 1.75, 20])

print("Series1:")

print(ps1)

print("\nSeries2:")

print(ps2)

# Compare the series using '==' and '!='

# Relational operators

print("\nCompare the elements of the two given Series:")

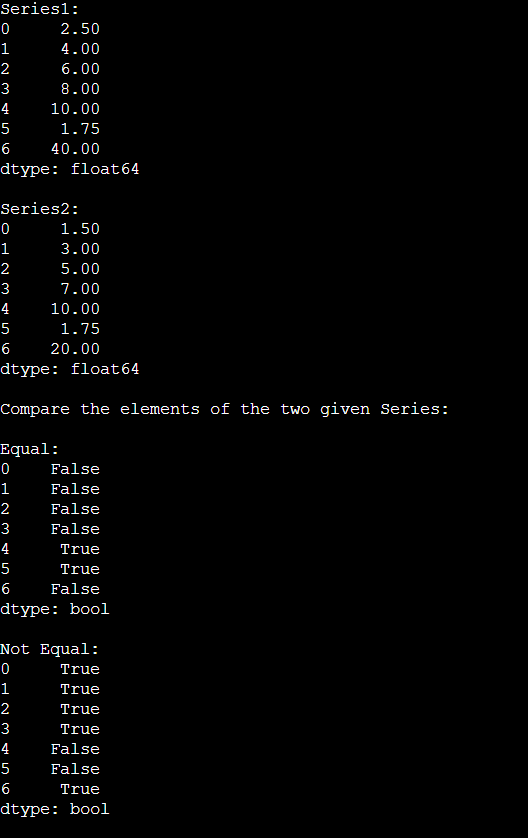
print("\nEqual:")

print(ps1 == ps2)

print("\nNot Equal:")

print(ps1 != ps2)

Output:



### 25. b. ii)

NumPy (Numerical Python) is a powerful library in Python used for numerical and scientific computing. It is especially well-suited for working with large, multi-dimensional arrays and matrices of numerical data.

ndarray :

The core of NumPy is the ndarray, which is a fast, memory-efficient, and flexible array object that allows you to store and manipulate large datasets.

Python’s default build-in list allows heterogeneous elements to be stored.

But in ndarray, you can only store homogeneous data type elements.

Eg:

arr = np.array([1, 'hello', 3.5])

print(arr)

Output: [‘1’,’hello’,’3.5]

Here, all elements are converted to string as it can’t store more than data type. Only single data type elements can be stored in this type of array.

NumPy arrays automatically convert mixed data types to the most general type to ensure homogeneity. In most cases, this results in the array being cast to a string type if there is any string in the data.