SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

2 – Vaishalinagar,
Nr. Amrapali Under Bridge Raiya Road,
Rajkot – 360001
Ph.No–(0281)2440478,2472590

3 – Vaishalinagar ,
Nr. Amrapali Under Bridge Raiya Road,
Rajkot - 360001
Ph.No–(0281)2471645

WHAT IS DATA STRUCTURE

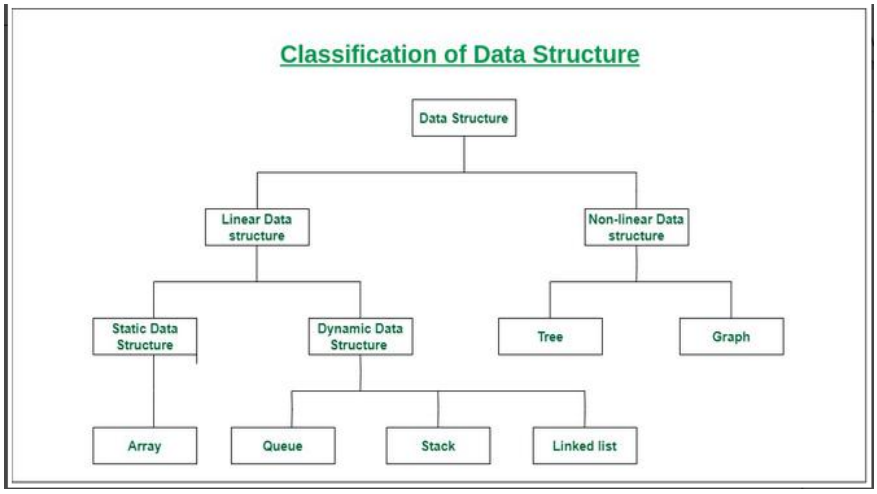
A **data structure** is a storage that is used to store and organize data. It is a way of arranging data on a computer so that it can be accessed and updated efficiently.

Linear Data Structures are a type of data structure in computer science where data elements are arranged **sequentially or linearly.** Each element has a previous and next adjacent ,except for the first and last elements.

Data structures where data elements are not arranged sequentially or linearly are called **non-linear data structures**. In a non-linear data structure, single level is not involved. Therefore, we can't traverse all the elements in single run only.

The analysis of algorithm.

- Time and space complexities.
- Asymptotic notation.
- Classes of algorithm.
- Big-Oh Notation
- Big-Omega Notation



SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.

(AFFILIATED TO SAURASHTRA UNIVERSITY)

2 – Vaishalinagar,
Nr. Amrapali Under Bridge Raiya Road,
Rajkot – 360001
Ph.No–(0281)2440478,2472590

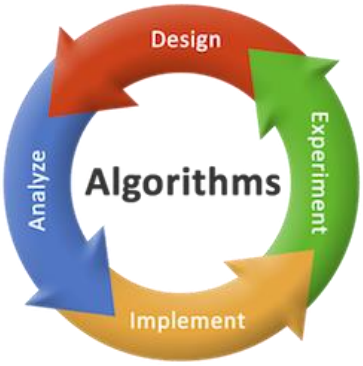
3 – Vaishalinagar ,
Nr. Amrapali Under Bridge Raiya Road,
Rajkot - 360001
Ph.No–(0281)2471645

Ch-1 (part 1)

Algorithm & Analysis

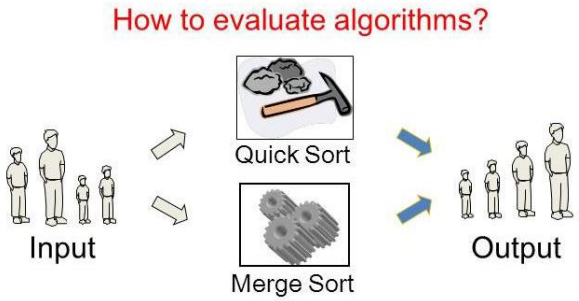
Algorithm analysis is an important part of **computational complexity theory**, which provides theoretical estimation for the required resources of an algorithm to solve a specific computational problem. Analysis of algorithms is the determination of the amount of **time** and **space** resources required to execute it.

1. **The analysis of algorithm.**



What Is Algorithm

An algorithm is a step-by-step procedure for solving a problem in a finite amount of time.



- Which one is better?
- What are the criteria?

Every algorithm should have the following five characteristics:

i. Input---The algorithm should take zero or more input.

ii. Output---The algorithm should produce one or more outputs.

iii. Definiteness---Each and every step of algorithm should be defined unambiguously.

iv. Effectiveness---A human should be able to calculate the values involved in the procedure of the algorithm using paper and pencil.

v. Termination---An algorithm must terminated after a finite number of steps.

2. Time and space Complexities.

The **time complexity** of a program/algorithm is the amount of computer time that it needs to run to completion.

The **space complexity** of a program/algorithm is the amount of memory that it needs to run to completion.

The exact time will depend on the implementation of the algorithm, programming language, optimizing capabilities of the compiler used and so on…

Some of the reasons for studying time complexity are:

- We may be interested to know in advance whether the program will provide a satisfactory real time response.
- There may be several possible solutions with different time requirement.

When we analyze an algorithm depends on the input data, there are three different types of time complexities which can be analyzed for an algorithm.

- Best case time complexity
- Average case time complexity
- Worst case time complexity

3. Asymptotic notation

- When we study algorithms, we are interested in characterizing them according to their efficiency.
- We are usually interesting in the order of growth of the running time of an algorithm, not in the **exact running time**. This is also referred to as the asymptotic notation.
- We need to develop a way to talk about **rate of growth of functions** so that we can compare algorithms.
- Asymptotic notation gives us a method for classifying functions according to their rate of growth.

SO, WHAT is the best solution to this problem?

- Measuring the actual running time is not particle at all.
- The running time generally depends on the size of the input.

Example:-

Array

4	1	2

3	4	1	2

- Add the new element in the array like 3

- Size of the input here in this case it depend on the size of the array.
- This clearly shows that the running time generally depends on the size of the input.
- There, if the size of the input is n, then f(n) is a function of n denotes the time complexity.

4. Classes of algorithm

- **Classification By implementation way**

An algorithm may be implemented according to different principles

1) Recursion or Iteration

A recursive algorithm means that it invokes itself repeatedly until a certain condition matches.

An iterative algorithms use repetitive constructs like loops and sometimes additional data structures like stacks to solve the given problems

2) Logical

This is the basic of the logic programming. In pure logic programming languages, the control component.

3) Serial or Parallel or Distributed

A computer which can execute on instruction of an algorithm at a time is known as serial computers. An algorithm designed for such an environment is called a serial algorithm.

4) Deterministic of Non-Deterministic

Deterministic algorithms solve the problem with exact decision at every step of the algorithm.

5) Exact or Approximate

While many algorithms reach an exact solution, approximation algorithms try to find an approximation that is close to the true solution.

- **By design paradigm**

There is a number of paradigms which is different from each other. Also, it will include many different types of algorithm.

1) **Brute-force or exhaustive search**

This is the natural method of trying every possible solution to see which is best.

2) **Divide and conquer**

A divide and conquer algorithm repeatedly reduces into smaller problems until the problems are not enough to solve easily.

3) **Dynamic programming**

When a problem shows optimal substructure, meaning the optimal solution to a problem can be constructed form optimal solutions to sub problems, and overlapping sub problems, meaning the same sub problems are used to solve many different problem instances, a quicker approach called dynamic programming avoids recomputing solutions that have already been computed

4) **The greedy method**

A greedy algorithm is similar to a dynamic programming algorithm, but the difference is that solutions to the sub problems do not have to be known at each stage, instead a "greedy"
choice can be made of what looks best for the moment.

5) **Linear programming**

When solving a problem using linear programming, specific inequalities involving the inputs are found and then an attempt is made to maximize some linear function of the inputs.

6) **Reduction**

This technique involves solving a difficult problem by transforming it into a better known problem for which asymptotically optimal algorithms

7) **Search and enumeration**

Many problems can be modeled as problems on graphs.
A graph exploration algorithm specifies rules for moving around a graph and is useful for such problems.
This category also includes search algorithms, branch and bound enumeration and backtracking

- **By field of study**

i. In the field of computer science has its own problem and requires efficient algorithm

ii. Related problems in one field are often studied together.

iii. Some example classes are search algorithms, sorting algorithms, merge algorithms etc…..

- **By Complexity**

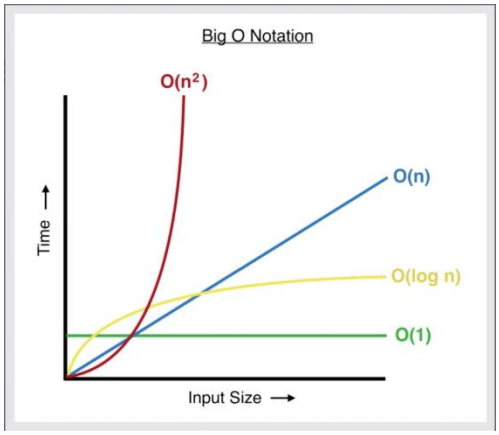
i. Algorithms can be classified by the amount of time they require to complete compared to their input size.

ii. There is a wide variety of some algorithms Complete in linear time

2 – Vaishalinagar,
Nr. Amrapali Under Bridge Raiya Road,
Rajkot – 360001
Ph.No–(0281)2440478,2472590

3 – Vaishalinagar ,
Nr. Amrapali Under Bridge Raiya Road,
Rajkot - 360001
Ph.No–(0281)2471645

5. Big-Oh Notation



This notation is known as the upper bound of an algorithm or worst case of an algorithm.

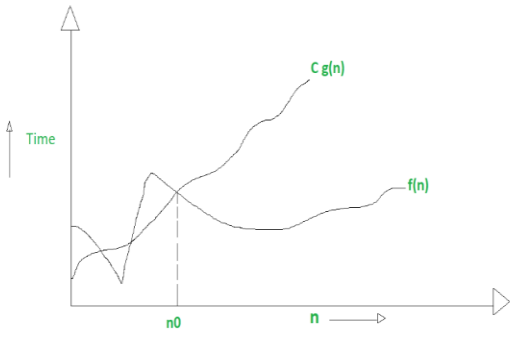
We can express algorithmic complexity using the big-O notation. For a problem of size N:

- A constant-time function/method is “order 1” : O(1)
- A linear-time function/method is “order N” : O(N)
- A quadratic-time function/method is “order N squared” : O(N^2)

Definition: Let g and f be functions from the set of natural numbers to itself. The function f is said to be O(g) (read big-oh of g), if there is a constant **c > 0** and a natural number $\mathbf{n_0}$ such that f(n) ≤ cg(n) for all n ≥ n_0 .
Note: O(g) is a set! Abuse of notation: f = O(g) does not mean f ∈ O(g).
The Big-O Asymptotic Notation gives us the Upper Bound Idea, mathematically described

6. Big-Omega Notation

It is define as lower bound and lower bound on an algorithm is the least amount of time required (the most efficient way possible, in other words best case).



Let **f(n)** define running time of an algorithm;
f(n) is said to be **Ω(g (n))** if there exists positive constant **C** and **(n0)** such that
0 <= Cg(n) <= f(n) for all n >= n0
n = used to given lower bound on a function
If a function is **Ω(n-square)** it is automatically **Ω(n)** as well.

SHREE H. N. SHUKLA COLLEGE OF I.T. & MGMT.
(AFFILIATED TO SAURASHTRA UNIVERSITY)
2 – Vaishalinagar,
Nr. Amrapali Under Bridge Raiya Road,
Rajkot – 360001
Ph.No–(0281)2440478,2472590
3 – Vaishalinagar ,
Nr. Amrapali Under Bridge Raiya Road,
Rajkot - 360001
Ph.No–(0281)2471645

Unit -1 Part-2 (File handling)

- **Concept of data files**

A data file is a computer file which stores data to be used by a computer application or system, including input and output data.

- **File handling**

In programming, we may require some specific input data to be generated several numbers of times. Sometimes, it is not enough to only display the data on the console. The data to be displayed may be very large, and only a limited amount of data can be displayed on the console, and since the memory is volatile, it is impossible to recover the programmatically generated data again and again. However, if we need to do so, we may store it onto the local file system which is volatile and can be accessed every time. Here, comes the need of file handling in C.
File handling in C enables us to create, update, read, and delete the files stored on the local file system through our C program. The following operations can be performed on a file.

- **Use of file handling functions**

No.	Function	Description
1	fopen()	open new or existing file
2	fprintf()	write data into the file
3	fscanf()	read data from the file
4	fputc()	write a character into the file
5	fgetc()	read a character from file
6	fclose()	close the file

7	fseek()	set the file pointer to given position
8	fputw()	write an integer to file
9	fgetw()	read an integer from file
10	ftell()	Return current position
11	rewind()	sets the file pointer to the beginning of the file

There are many functions in the C library to open, read, write, search and close the file. A list of file functions are given below:

1) **Fopen()**
   - Opening File: fopen()

We must open a file before it can be read, write, or update. The fopen() function is used to open a file. The syntax of the fopen() is given below.

Syntax:- **FILE** *fopen(" filename.txt ","w");

The fopen() function accepts two parameters:

- The file name (string). If the file is stored at some specific location, then we must mention the path at which the file is stored. For example, a file name can be like **"c://file_program/file.txt"**.
- The mode in which the file is to be opened. It is a string.

We can use one of the following modes in the fopen() function.

Mode	Description
r	opens a text file in read mode
w	opens a text file in write mode
a	opens a text file in append mode
r+	opens a text file in read and write mode
w+	opens a text file in read and write mode
a+	opens a text file in read and write mode
rb	opens a binary file in read mode
wb	opens a binary file in write mode
ab	opens a binary file in append mode
rb+	opens a binary file in read and write mode
wb+	opens a binary file in read and write mode
ab+	opens a binary file in read and write mode

Example 1:- Definition - Create a file with file-function

```
#include<stdio.h>
#include<conio.h>
Void main()
{
        FILE  *fptr;
        // Create a file
        fptr = fopen("file.txt", "w");
        // Close the file
        fclose(fptr); getch(); }
```

2) Fclose()

- *Closing the file*

Did you notice the fclose() function in our example above?

This will close the file when we are done with it.

It is considered as good practice, because it makes sure that:

- Changes are saved properly
- Other programs can use the file (if you want)
- Clean up unnecessary memory space

3) **Fprintf()**

- *Write To a File*

Let's use the w mode from the previous chapter again, and write something to the file we just created.

The w mode means that the file is opened for **writing**. To insert content to it, you can use the fprintf() function and add the pointer variable (fptr in our example) and some text:

Example :-2 Create a filefunction fprintf()

```
FILE *fptr;

// Open a file in writing mode
fptr = fopen("file.txt", "w");

// Write some text to the file
fprintf(fptr, "Some text");

// Close the file
fclose(fptr);
```

Note: If you write to a file that already exists, the old content is deleted, and the new content is inserted. This is important to know, as you might accidentally erase existing content.

- *Append Content To a File*

If you want to add content to a file without deleting the old content, you can use the a mode.

The a mode appends content at the end of the file:

Example :-3

Create a file with appends mode.

```
FILE *fptr;

// Open a file in append mode
fptr = fopen("filename.txt", "a");

// Append some text to the file
fprintf(fptr, "\nHi everybody!");

// Close the file
fclose(fptr);
```

4) Fscanf()

C fscanf() function reads formatted input from a file. This function is implemented in file-related programs for reading formatted data from any file specified in the program.

Syntax:- int fscanf(FILE *stream, const char *format, ...)

Example:-4

Store the employee details in file.

```
#include<conio.h>
#include<stdio.h>
void main()
{
int eno,salary;
char ename[20];
FILE *fptr;
clrscr();
0fptr=fopen(“write.txt”,”w”);
printf("Enter Employee No:-") ;
scanf("%d",&eno);
printf("enter Employee Name:-");
scanf("%s",&ename);
printf("Enter Employee Salary:-");
scanf("%d",&s+ alary);
fprintf(fptr,"%d",eno);
fprintf(fptr,"%s",ename);
fprintf(fptr,"%d",salary);
fclose(fptr);
getch();

}
```

Example :-5

Show the Employee Details on the screen.

```
#include<conio.h>
#include<stdio.h>
void main()
{
   int eno,salary;
char ename[20];
file *fptr;
clrscr();
fprt = fopen(“writer.txt”,”r”);
fscanf(fptr,”%d”,&eno);
fscanf(fptr,”%s”,&ename);
fscanf(fptr,“%d”,&salary);
printf(“the data on file to your program\n”);
printf(“\n employee no is :- %d”,eno);
printf(“\n employee name is :- %s”, ename);
printf(“\n employee salary is :- %d”, salary);
fclose(fptr);
getch();
}
```

5)getw() & 6) putw()

C getw() function is used to read an integer from a file that has been opened in read mode. It is a file handling function that is used for reading integer values.

Syntax :- int getw (FILE *fp);

C putw() function is used to write an integer in a file that has been written in Write mode. It is a file handling function that is used for writing integer values

Systax :- putw (int num, FILE *fp);

Example -6

create the integer values us to putw Function.

```
#include<stdio.h>
#include<conio.h>
void main( )
{
  FILE *fp;
 int i;
   fp = fopen ("num.txt", "w");
   for (i =1; i<= 10; i++)
{
 putw (i, fp);

 }
```

Example :-7

Show the integer values on the screen.

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE *fp;
  fp =fopen ("num.txt", "w");
 printf ("file content is\n");
 for (i =1; i<= 10; i++)
{
   i= getw(fp);
 printf("%d",i);
  printf("\n");
 }

 fclose (fp);
```

```
 fclose (fp);
getch();
}
```

7)Fseek ()

fseek() is used to move the file pointer associated with a given file to a specific position.

Syntax:-

int **fseek**(FILE **pointer*, long int *offset*, int *position*);

Parameters

- **pointer:** It is the pointer to a FILE object that identifies the stream.
- **offset:** It is the number of bytes to offset from the position
- **position:** It is the position from where the offset is added. Position defines the point with respect to which the file pointer needs to be moved. It has three values:
    - **SEEK_END**: It denotes the end of the file.
    - **SEEK_SET**: It denotes starting of the file.
    - **SEEK_CUR**: It denotes the file pointer's current position.

Example :- 8

```
#include <stdio.h>
int main()
{
   FILE* fp;
   fp = fopen("file.txt", "r");

   // Moving pointer to end
   fseek(fp, 0, SEEK_END);
```

```
    // Printing position of pointer
    printf("%ld", ftell(fp));
    return 0;
}
```

8) Ftell()

C language, ftell() returns the current file position of the specified stream with respect to the starting of the file. This function is used to get the total size of file after moving the file pointer at the end of the file. It returns the current position in long type and file can have more than 32767 bytes of data.

Syntax :- long int ftell(FILE *stream)

Example :-9

```
#include <stdio.h>
#include<conio.h>
void main () {
  FILE *f;
  int len;
  f = fopen("one.txt", "r");
  if(f == NULL) {
    perror("Error opening file");
    return(-1);
  }
  fseek(f, 0, SEEK_END);
  len = ftell(f);
  fclose(f);
  printf("Size of file: %d bytes", len);
  getch();
}
```

9) Rewind

The rewind() function sets the file pointer at the beginning of the stream. It is useful if you have to use stream many times.

Syntax :- `void rewind(FILE *stream)`

Example :-10

```
int main()
{
        FILE* fp = fopen("test.txt", "r");
        if (fp == NULL) {/* Handle open error */}
        rewind(fp);
        return 0;
}
```

10) freopen

The freopen() function is used to close the file that is currently associated with stream and reassigns stream to the file that is specified by filename. With the freopen() function, a new file associated with stream is opened with the given mode, which is a character string specifying the type of access requested.

Syntax :- `FILE *freopen(const char *filename, const char *mode, FILE *stream)`

Example :-11

```
#include <stdio.h>
int main () {
  FILE *fp;
  printf("This text is redirected to stdout\n");
  fp = freopen("file.txt", "w+", stdout);
  printf("This text is redirected to file.txt\n");
  fclose(fp);
  return(0);
}
```

11) remove

The **remove()** function in C/C++ can be used to delete a file. The function returns 0 if the file is deleted successfully, Otherwise, it returns a non-zero value. The remove() is defined inside the **<stdio.h>** header file.

Syntax :- **remove**("*filename*");

Example :-12

```
#include <stdio.h>
int main()
{
   if (remove("abc.txt") == 0)
      printf("Deleted successfully");
   else
      printf("Unable to delete the file");
   return 0;
}
```

12) rename

The **rename()** function is used to rename a file in C. It changes the name of the file from **old_name** to **new_name** without modifying the content present in the file.

Syntax :- int **rename** (const char **old_name*, const char **new_name*);

EXAMPLE:-13

```
// C program to demonstrate use of rename()
#include <stdio.h>
int main()
{
  // Old file name
  char old_name[] = "FILE.txt";
  // Any string
  char new_name[] = "RENAME.txt";
  int value;
```

```
        // File name is changed here
        value = rename(old_name, new_name);
        // Print the result
        if (!value) {
                printf("%s", "File name changed successfully");
        }
        else {
                perror("Error");
        }
        return 0;
      }
```

13) Feof(file end of file)

C feof() function is used to determine if the end of the file (stream) specified has been reached or not. This function keeps on searching the end of the file (EOF) in your file program.

Syntax :- int feof(FILE *stream)

Example :-14

```
#include<stdio.h>
int main()
{
   FILE *f = NULL;
   char buf[50];
   filee = fopen("infor.txt","r");
   if(filee)
   {
      while(!feof(filee))
      {
         fgets(buf, sizeof(buf), filee);
         puts(buf);
      }
      fclose(filee);
   }
   return 0;
}
```

14) Ferror

The C library function **int ferror(FILE *stream)** tests the error indicator for the given stream.

Syntax :- int ferror(FILE *stream)

Example :-15

```
#include <errno.h>
#include <stdio.h>

int main()
{

   FILE* fp;

   fp = fopen("GeeksForGeeks.txt", "r");

   printf("Value of errno: %d\n", errno);

   return 0;

}
```

2 – Vaishalinagar,
Nr. Amrapali Under Bridge Raiya Road,
Rajkot – 360001
Ph.No–(0281)2440478,2472590

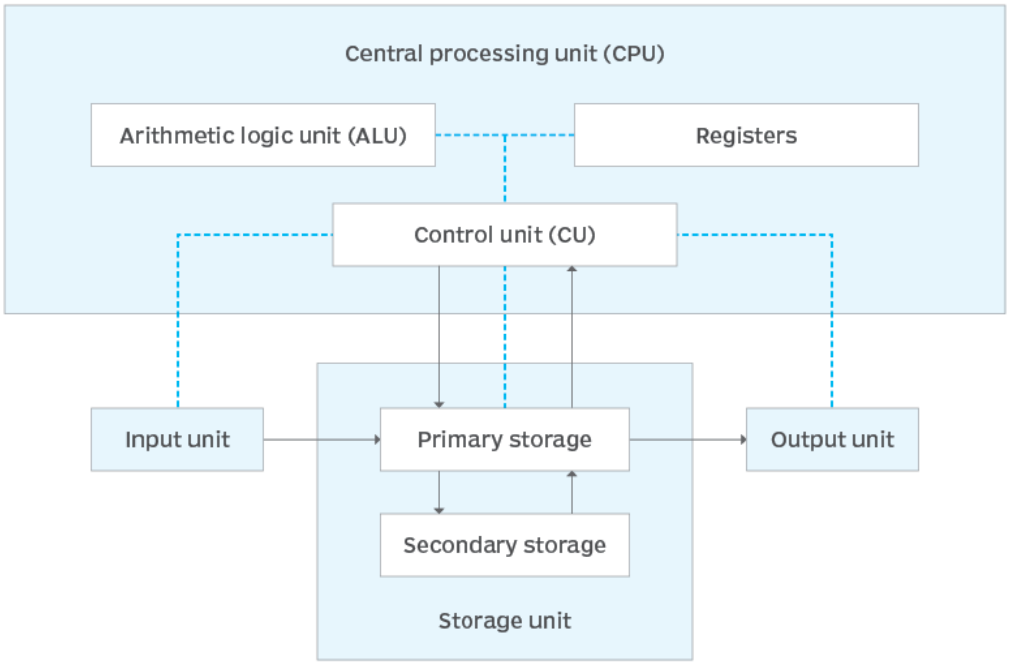
3 – Vaishalinagar ,
Nr. Amrapali Under Bridge Raiya Road,
Rajkot - 360001
Ph.No–(0281)2471645

I/O operations

The **DMA mode (**Direct Memory Access) of data transfer reduces the CPU's overhead in handling I/O operations. It also allows parallelism in CPU and I/O operations. Such parallelism is necessary to avoid the wastage of valuable CPU time while handling I/O devices whose speeds are much slower as compared to CPU. The concept of DMA operation can be extended to relieve the CPU further from getting involved with the execution of I/O operations. This gives rise to the development of special purpose processors called **Input-Output Processor (IOP) or IO channels**.

The Input-Output Processor (IOP) is just like a CPU that handles the details of I/O operations. The IOP can fetch and execute its own instructions that are specifically designed to characterize I/O transfers. In addition to the I/O-related tasks, it can perform other processing tasks like arithmetic, logic, branching, and code translation. The main memory unit takes a pivotal role. It communicates with the processor by means of DMA.

The Input-Output Processor is a specialized processor which loads and stores data in memory along with the execution of I/O instructions. It acts as an interface between the system and devices. It involves a sequence of events to execute I/O operations and then store the results in memory.



Input-Output Processor

Features of an Input-Output Processor

- **Specialized Hardware:** An IOP is equipped with specialized hardware that is optimized for handling input/output operations. This hardware includes input/output ports, DMA controllers, and interrupt controllers.
- **DMA Capability:** An IOP has the capability to perform Direct Memory Access (DMA) operations. DMA allows dat a to be transferred directly between peripheral devices and memory without going through the CPU, thereby freeing up the CPU for other tasks.
- **Protocol Handling:** An IOP can handle communication protocols for different types of devices such as Ethernet, USB, and SCSI(Small Computer System Interface). This allows the IOP to interface with a wide range of devices without requiring additional software support from the CPU.
- **Buffering:** An IOP can buffer data between the CPU and peripheral devices. This allows the IOP to handle large amounts of data without overloading the CPU or the peripheral devices.
- **Command Processing:** An IOP can process commands from peripheral devices independently of the CPU. This allows the CPU to focus on executing application programs while the IOP handles peripheral device commands.

- **Parallel Processing:** An IOP can perform input/output operations in parallel with the CPU. This allows the system to handle multiple tasks simultaneously and improve overall system performance.

Applications ; of I/O Processors

- **Industrial Control Systems:** I/O processors can be used in industrial control systems to interface with various control devices and sensors. The I/O processor can provide precise timing and control signals, and can also perform local processing of the input data.
- **Multimedia Applications:** I/O processors can be used in multimedia applications to handle the input and output of multimedia data, such as audio and video. The I/O processor can perform real-time processing of multimedia data, including decoding, encoding, and compression.
- **Network Communication Systems:** I/O processors can be used in network communication systems to handle the input and output of data packets. The I/O processor can perform packet routing, filtering, and processing, and can also perform encryption and decryption of the data.
- **Storage Systems:** I/O processors can be used in storage systems to handle the input and output of data to and from storage devices. The I/O processor can handle high-speed data transfer and perform data caching and prefetching operations.

Advantages of Input-Output Processor

- The I/O devices can directly access the main memory without the intervention of the processor in I/O processor-based systems.

- **Reduced Processor Workload:** With an I/O processor, the main processor doesn't have to deal with I/O operations, allowing it to focus on other tasks. This results in more efficient use of the processor's resources and can lead to faster overall system performance.
- **Improved Data Transfer Rates:** Since the I/O processor can access memory directly, data transfers between I/O devices and memory can be faster and more efficient than with other methods.

- **Increased System Reliability:** By offloading I/O tasks to a dedicated processor, the system can be made more fault-tolerant. For example, if an I/O operation fails, it won't affect other system processes.
- **Scalability:** I/O processor-based systems can be designed to scale easily, allowing for additional I/O processors to be added as needed. This can be particularly useful in large-scale data centres or other environments where the number of I/O devices is constantly changing.
- **Flexibility:** I/O processor-based systems can be designed to handle a wide range of I/O devices and interfaces, providing more flexibility in system design and allowing for better customization to meet specific requirements.

Disadvantages of Input-Output Processor

- **Cost:** I/O processors can add significant costs to a system due to the additional hardware and complexity required. This can be a barrier to adoption, especially for smaller systems.
- **Increased Complexity:** The addition of an I/O processor can increase the overall complexity of a system, making it more difficult to design, build, and maintain. This can also make it harder to diagnose and troubleshoot issues.
- **Synchronization Issues:** With multiple processors accessing the same memory, synchronization issues can arise, leading to potential data corruption or other errors.
- **Lack of Standardization:** There are many different I/O processor architectures and interfaces available, which can make it difficult to develop standardized software and hardware solutions. This can limit interoperability and make it harder for vendors to develop compatible products.

Command line arguments

The most important function of C is the main() function. It is mostly defined with a return type of int and without parameters.

```
int main() {

   ...

}
```

We can also give command-line arguments in C. Command-line arguments are the values given after the name of the program in the command-line shell of Operating Systems. Command-line arguments are handled by the main() function of a C program.

To pass command-line arguments, we typically define main() with two arguments: the first argument is the **number of command-line arguments** and the second is a **list of command-line arguments.**

Syntax

```
int main(int argc, char *argv[]) { /* ... */ }
```

or

```
int main(int argc, char **argv) { /* ... */ }
```

Here,

0 seconds of 0 secondsVolume 0%

- **argc (ARGument Count)** is an integer variable that stores the number of command-line arguments passed by the user including the name of the program. So if we pass a value to a program, the value of argc would be 2 (one for argument and one for program name)
- The value of argc should be non-negative.
- **argv (ARGument Vector)** is an array of character pointers listing all the arguments.
- If argc is greater than zero, the array elements from argv[0] to argv[argc-1] will contain pointers to strings.
- argv[0] is the name of the program , After that till argv[argc-1] every element is command -line arguments.

For better understanding run this code on your Linux machine.

Example

The below example illustrates the printing of command line arguments.

```
// C program named mainreturn.c to demonstrate the working
// of command line arguement
#include <stdio.h>
// defining main with arguments
int main(int argc, char* argv[])
{   printf("You have entered %d arguments:\n", argc);
   for (int i = 0; i < argc; i++) {
      printf("%s\n", argv[i]);
   }   return 0;}
```

Output

You have entered 4 arguments:

./main

Properties of Command Line Arguments in C

1. They are passed to the main() function.
2. They are parameters/arguments supplied to the program when it is invoked.
3. They are used to control programs from outside instead of hard coding those values inside the code.
4. argv[argc] is a NULL pointer.
5. argv[0] holds the name of the program.
6. argv[1] points to the first command line argument and argv[argc-1] points to the last argument.

Note: *You pass all the command line arguments separated by a space, but if the argument itself has a space, then you can pass such arguments by putting them inside double quotes “” or single quotes ”.*

2 – Vaishalinagar,
Nr. Amrapali Under Bridge Raiya Road,
Rajkot – 360001
Ph.No–(0281)2440478,2472590

3 – Vaishalinagar ,
Nr. Amrapali Under Bridge Raiya Road,
Rajkot - 360001
Ph.No–(0281)2471645

Output in different scenarios:

1. Without argument:

2. Three arguments:

3. Single Argument:

4. A single argument in quotes separated by space