FortuneCloud
We Understand Technology

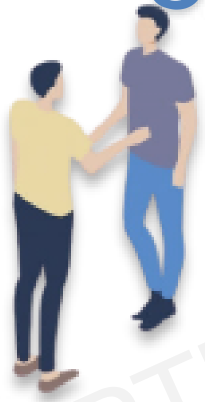
C Language - eBook

Fortune Cloud Technologies Group
2nd Floor, Abhinav Apartment, Beside Congress Bhavan, Shivaji Nagar, Pune - 411005
Landmark: Near Pune Municipal Corporation (Ma.na.pa) Bus Stand, Pune
Contact No: 9766439090 / 7083777567
Website: www.fortunecloudindia.com

C-Language

FortuneCloud
We Understand Technology

→ Language



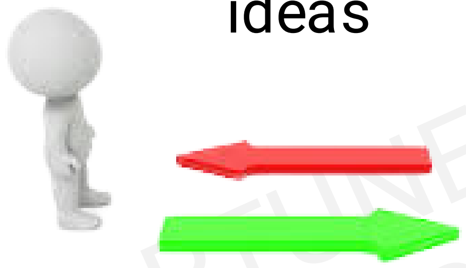
Set of instructions we exchange or give to someone in a particular language such as English, Marathi, Hindi etc.

→ Computer Language

It's a predefined application , used to communicate with the computers,.

Need of Computer Languages

Sharing or exchanging ideas

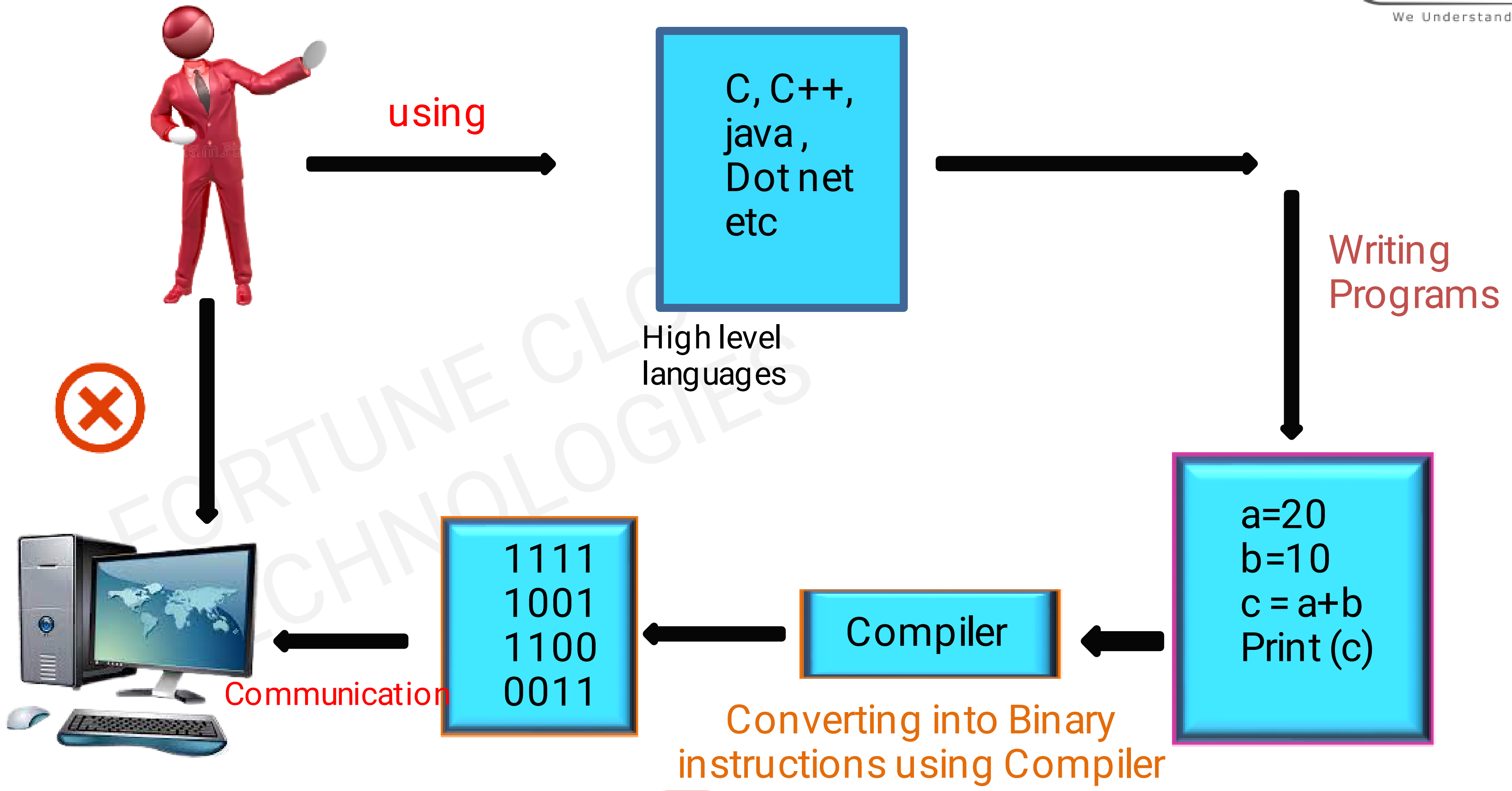


Need particular language such as Eng, Hindi , Marathi etc

Communicate with computer

Need machine/computer understandable language. Binary Language but we can not pass instructions in Binary code. So we need to learn a particular computer languages. Such as c, c++, java etc.

Process of Communication with computer



Computer-Languages

Using Computer Languages we are developing Applications

Standalone Application

-Must be installed.
Ex-Chrome, VLC media player, Ms-Office etc
-Compatible for single operative system

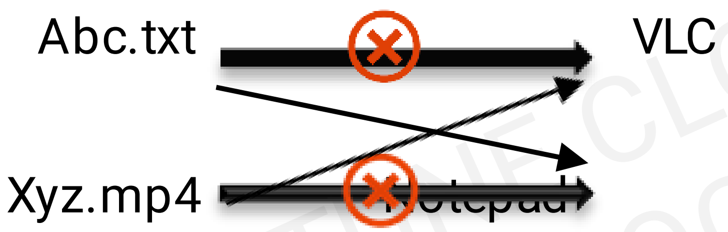
Web Application

-No need to install.
Ex- FB , Gmail its not necessary to install i
computer
-Independent to operative system

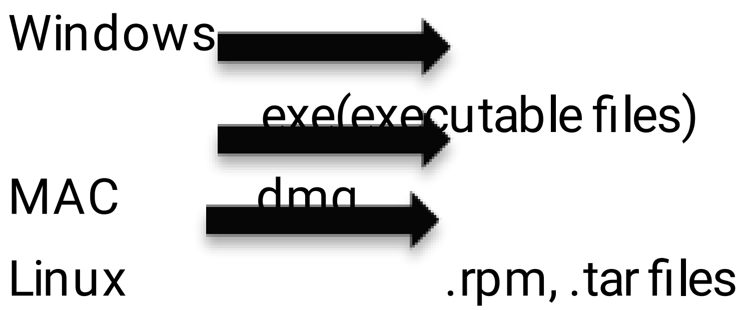
Why the Operating system Understand only a particular Software or an application

Operating system Understand only a particular Software or an application
Reason…….. We should understand File – Extensions.

File-Extensions



Os-Extensions



Programming Languages are Standalone applications or web applications??

Any programming Languages u can take… all are Standalone Applications…!

If you say that, you want to write a Java Program…. For that you must install Java

C C++	Platform Dependent
Java .net Android	Platform independent

Platform Dependency

Why C Language can run only on single platform?

Depends on Operating system We download C Language

WINDOWS Compiler work for WINDOWS OS

WINDOWS → C.exe = Compiler+Library

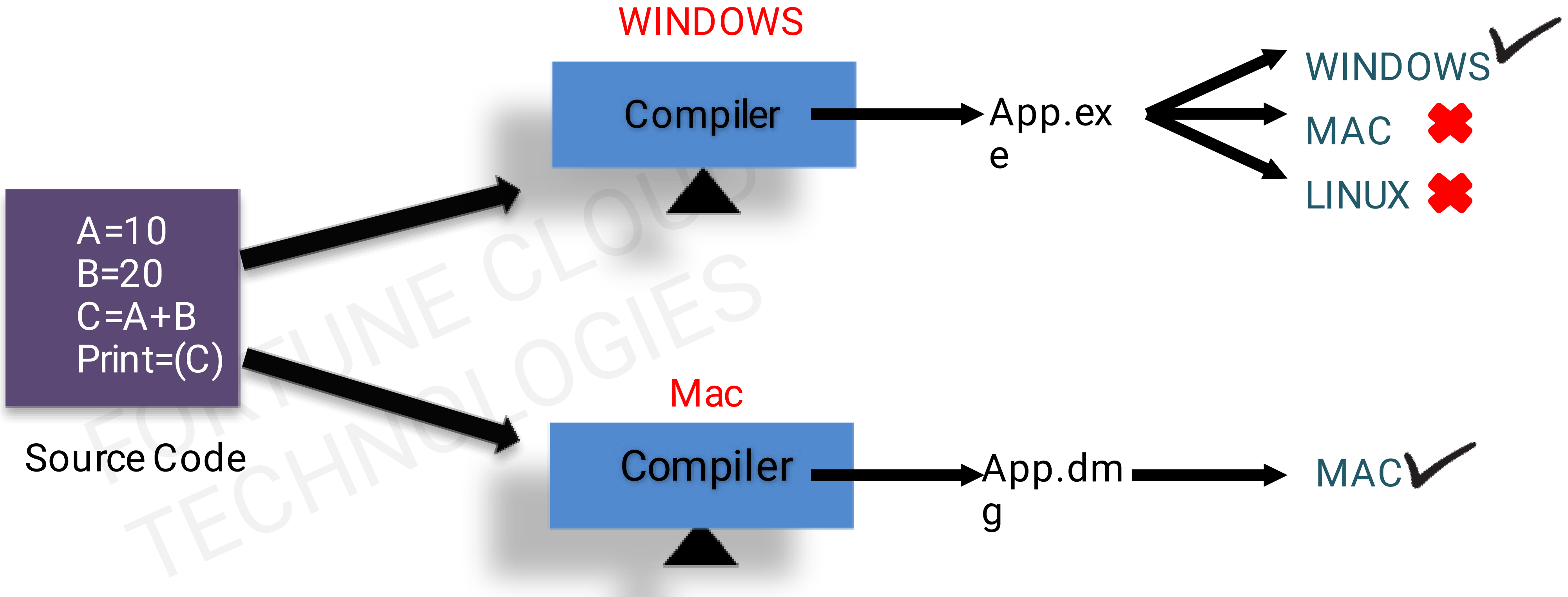
Along with C.exe we have to install windows compile and liabrary

MaC Compiler work for Mac OS

Mac → C.dmg = Compiler+Library

Along with C.dmg we have to install Mac compile and liabrary

Platform Dependency



C Applications run on a specific platform because of Compiler's dependency. On which or on which platform we compile the application, we can run the application on the same platform.

Library and IDE

Library is a collection of header files. Every header file includes related predefined things .(predefined functions and predefined variables)

C Library

stdio.h	conio.h	string.h

printf();
scanf();
	clrscr();
	getch();
		strlen();
		strrev();

graphics.h	**math.h**	**dos.h**

setcolor();
bkcolor();
	rand();
	pow();
		getdate();
		gettime();

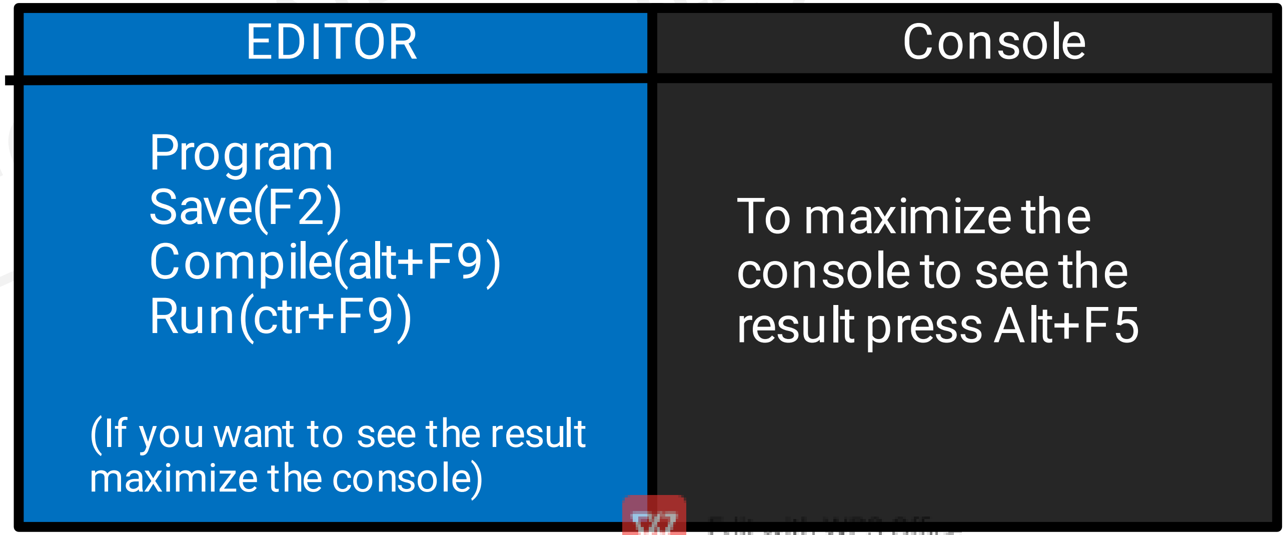
IDE:Integrated Development Environment

If you want to execute any program , a particular environment setup is required.
Ex:c, c++ programs executes in the original environment of operating system either Mac, windows, Linux

$$C,c++ \longrightarrow \begin{array}{l}\text{WINDOWS}\\ \text{MAC}\\ \text{LINUX}\end{array}$$

Note:As a basic programmer its not possible thing to work with a direct environment.. So they introduced IDE.the best example fOr IDE is **Blue screen**

BLUE SCREEN

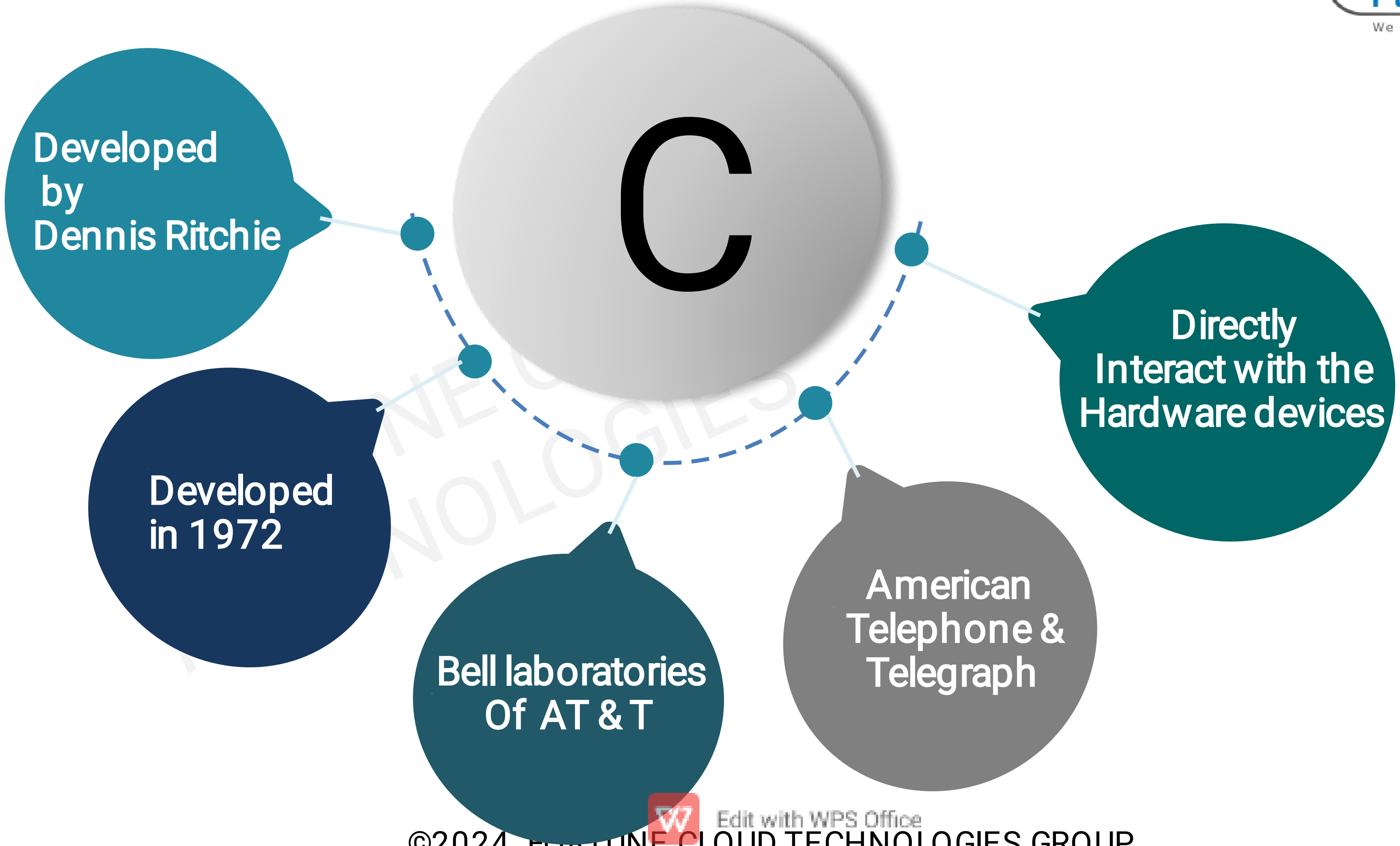


EDITOR	Console

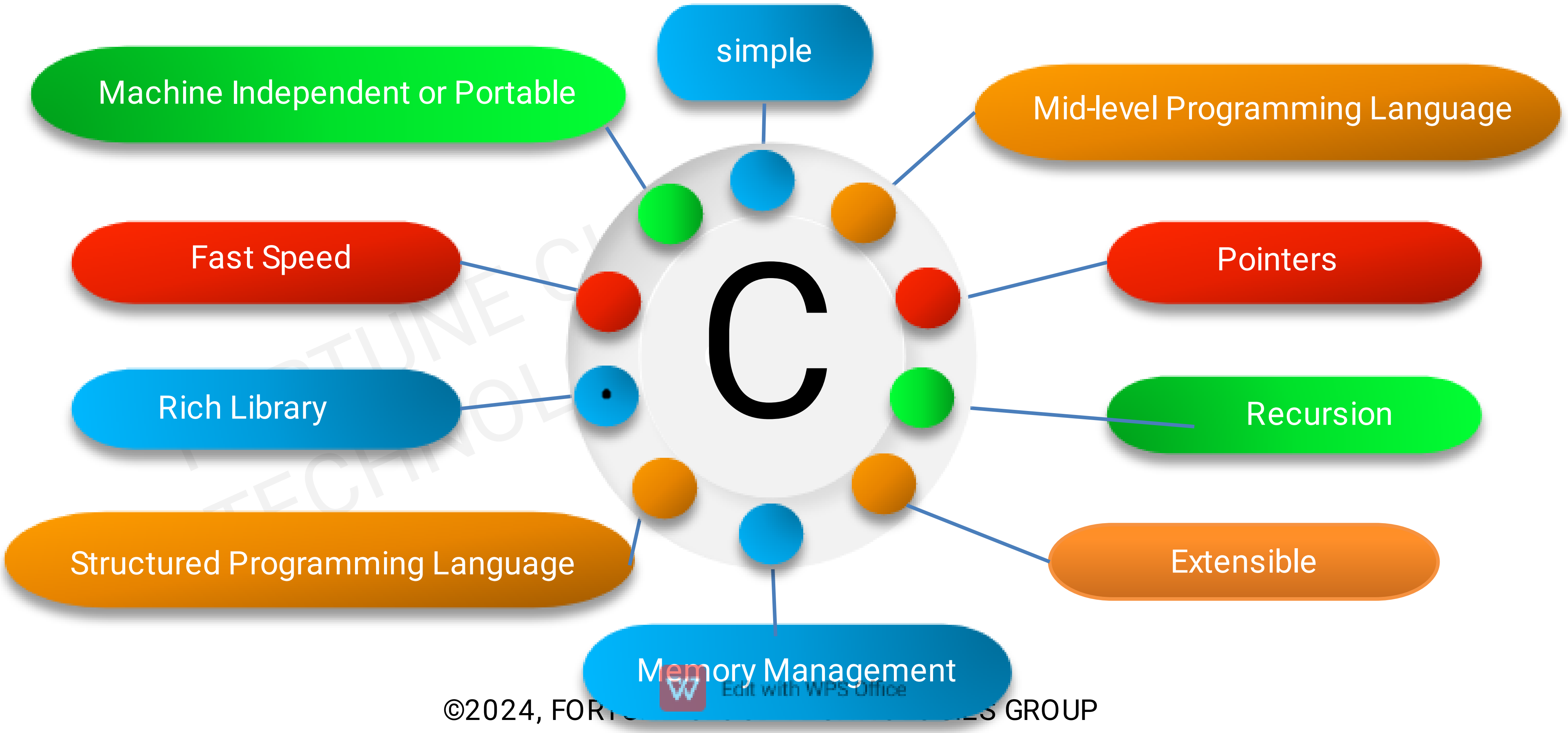
Program
Save(F2)
Compile(alt+F9)
Run(ctr+F9)
 	
(If you want to see the result maximize the console)
	To maximize the console to see the result press Alt+F5

C Introduction

History

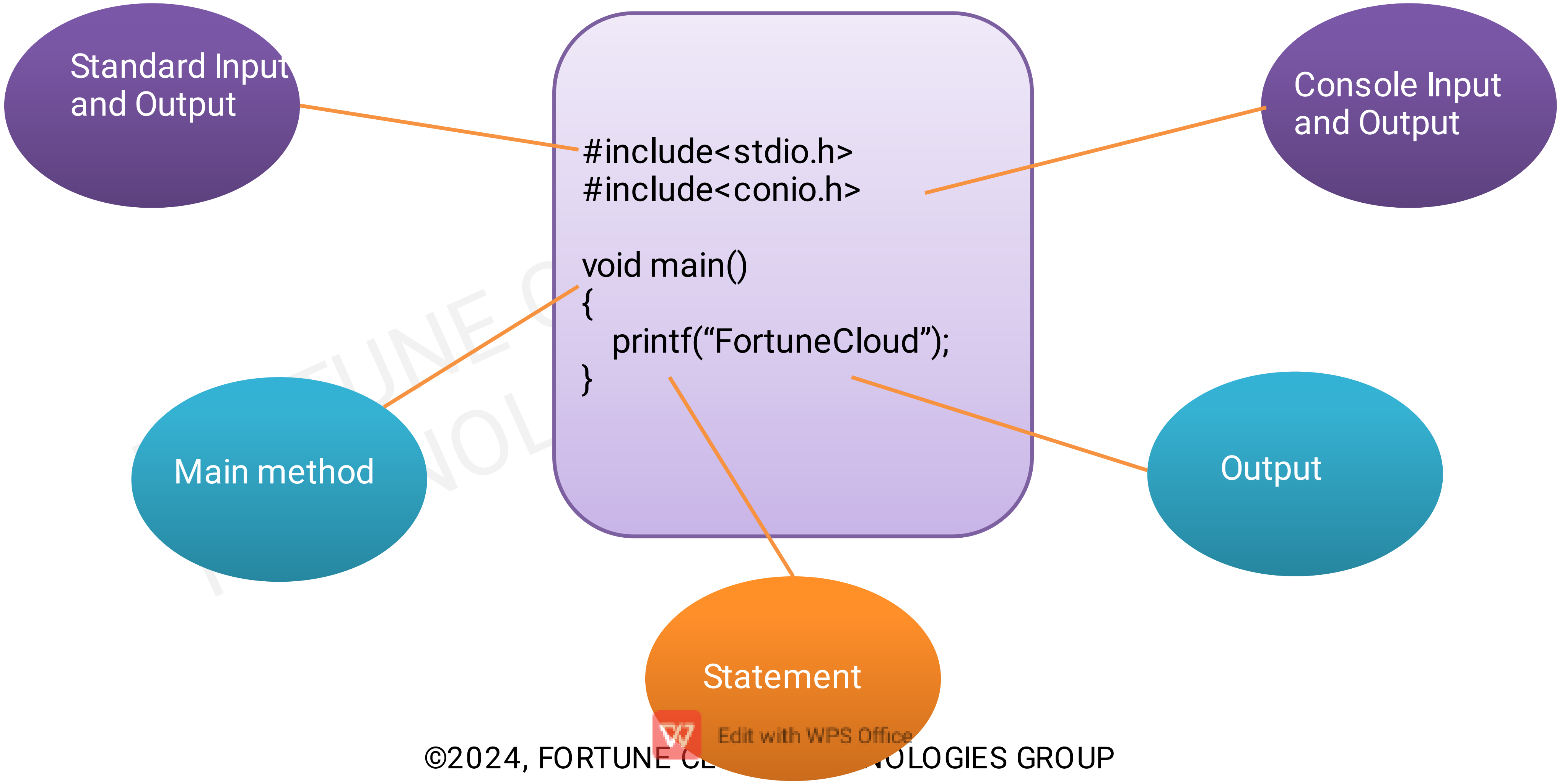

C
Developed by Dennis Ritchie
Developed in 1972
Bell laboratories Of AT & T
American Telephone & Telegraph
Directly Interact with the Hardware devices

Features


simple
Machine Independent or Portable
Mid-level Programming Language
Fast Speed
C
Pointers
Rich Library
Recursion
Structured Programming Language
Extensible
Memory Management

C Installation

Introduction to first program



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

void main()
{
   printf("FortuneCloud");
}
```

Variables

- A variable is a name of the memory location.
- A variable is nothing but a name given to a storage area.
- The name of a variable can be composed of letters, digits, and the underscore character.

Rules for defining variables

- A variable can have alphabets, digits, and underscore.
- A variable name can start with the alphabet, and underscore only.
- It can't start with a digit.
- No whitespace is allowed within the variable name.
- A variable name must not be any reserved word or keyword, e.g. int, float, etc.

Valid Variable Names

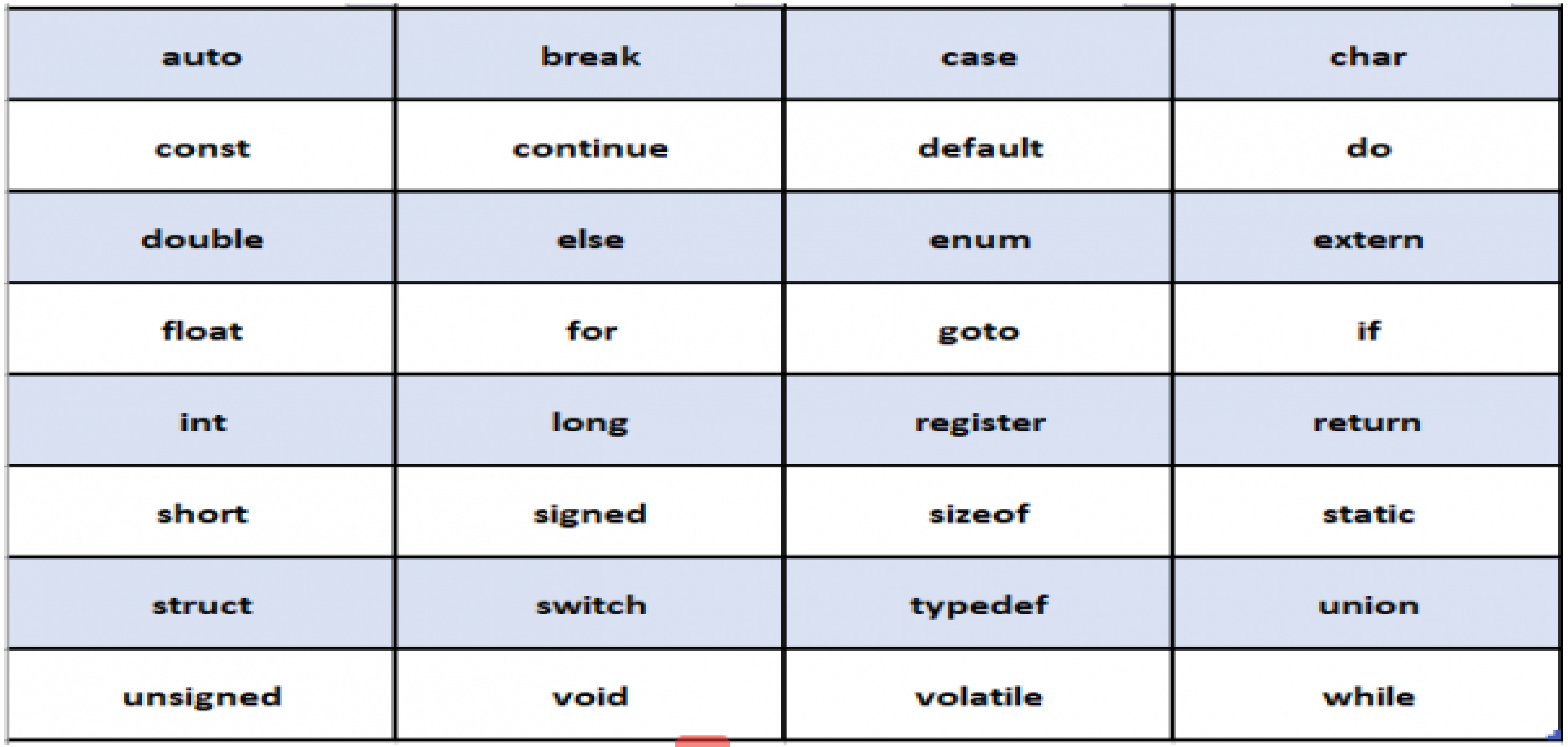
```
int a;
int _ab;
int a30;
```

Invalid Variable names

```
int 2;
int a b;
int long;
```

Keywords

C language has **32** reserved words as per ANSI standards.



auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while

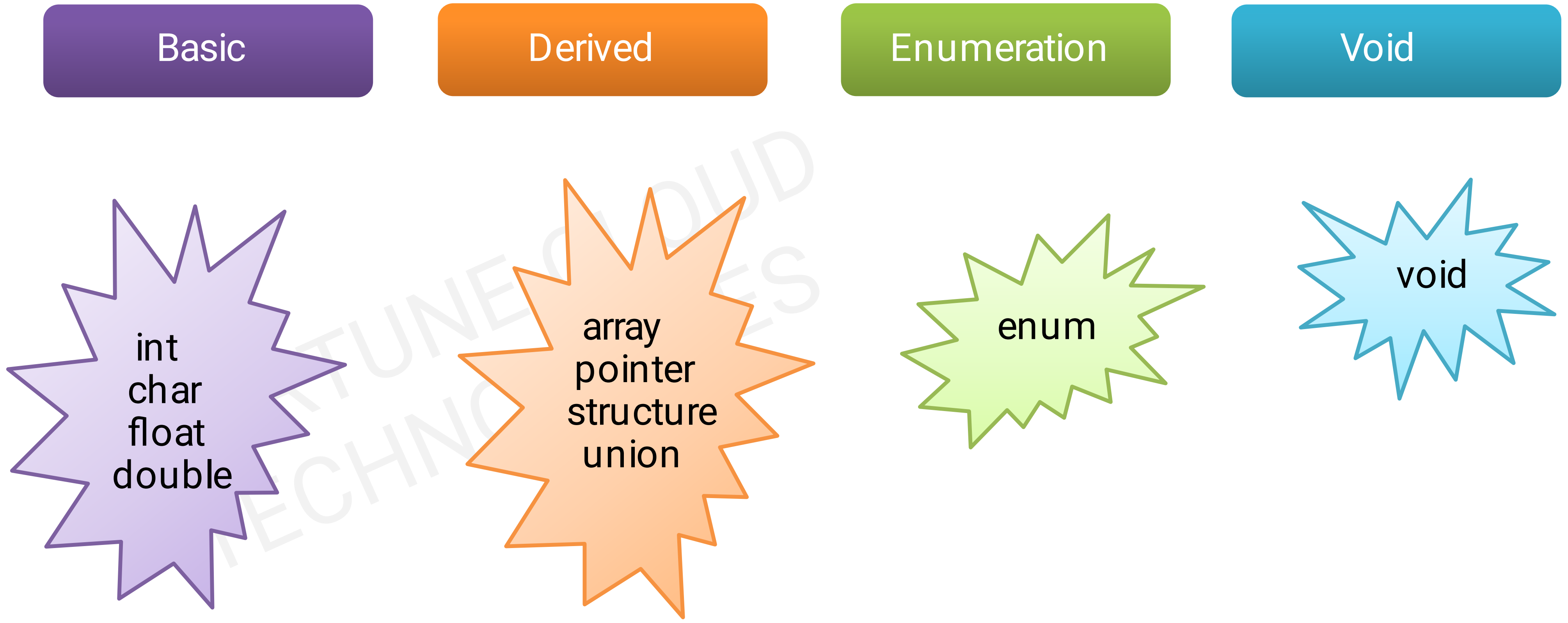
Datatypes

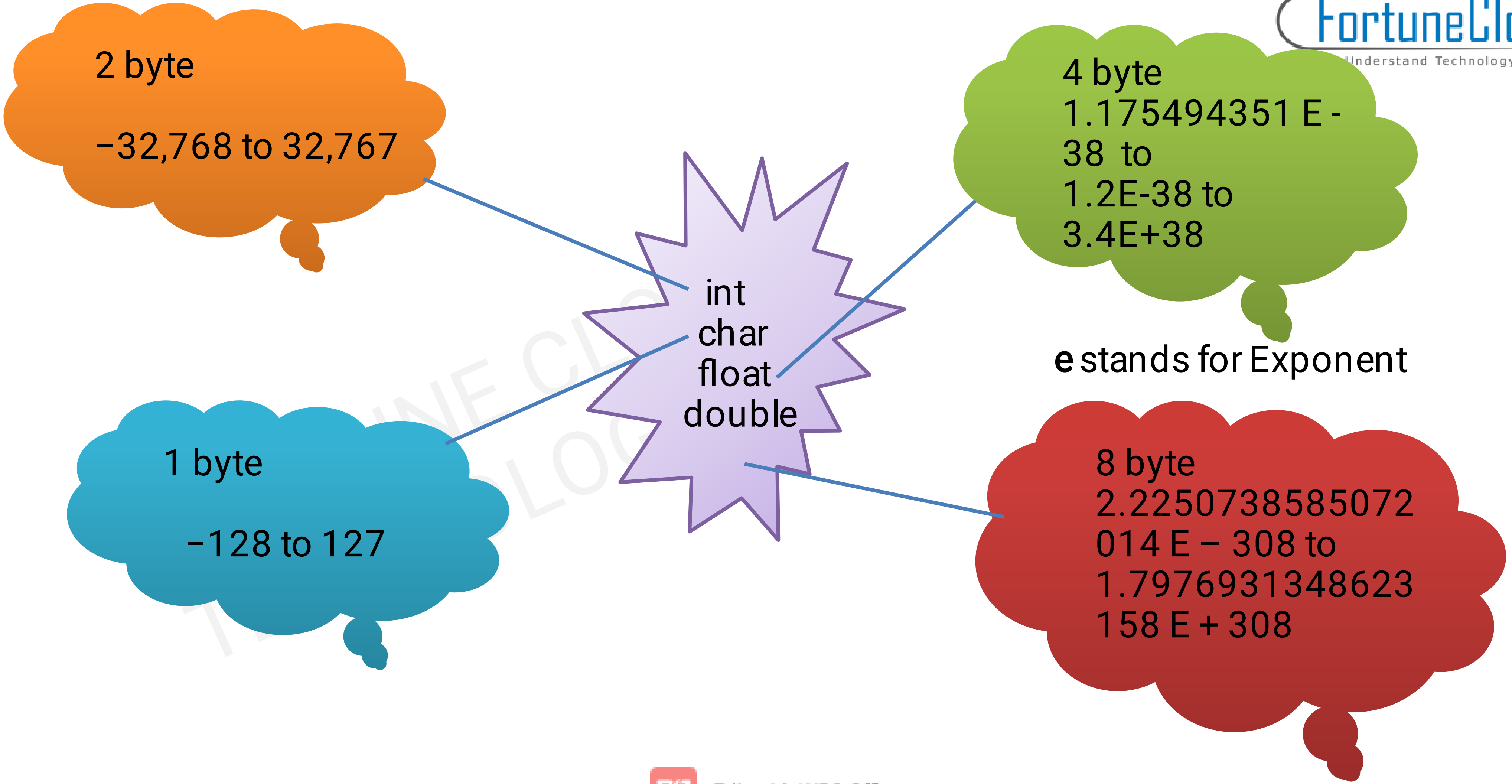
extensive system used for declaring variables or functions of different types

The type of a variable determines how much space it occupies in storage

It specifies the type of data that a variable can store such as integer, floating, character, etc.

Datatypes in C





e stands for Exponent

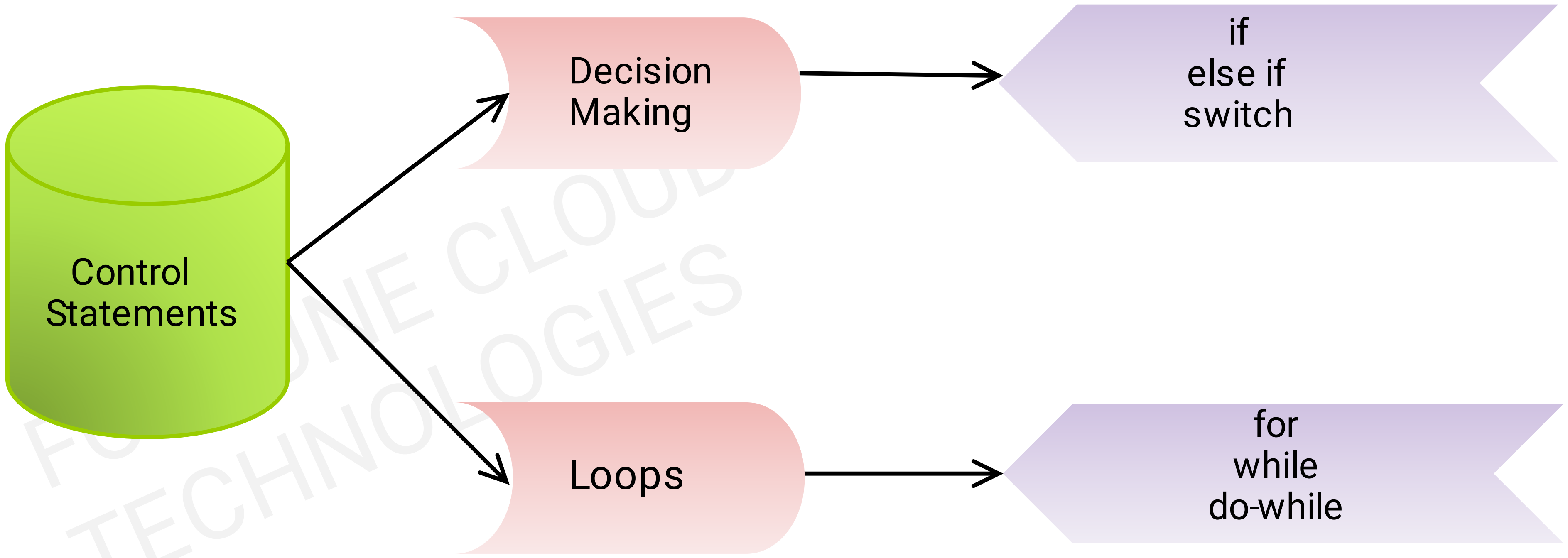
Comparison Operators

=	a=b
==	a==b
!=	a!=b
>	a>b
<	a<b
>=	a>=b
<=	a<=b
!<	a!<b
!>	a!>b

Arithmetic Operators

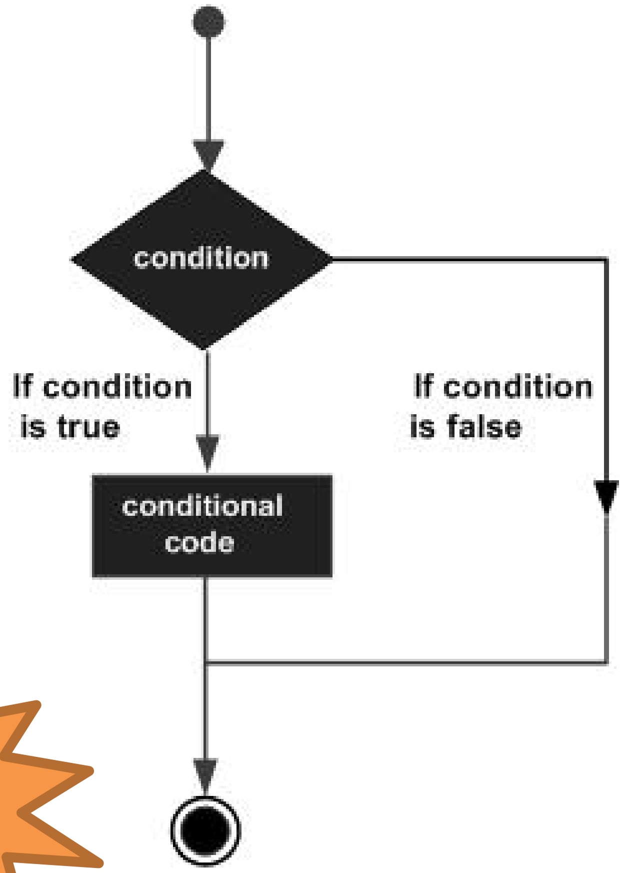
+	a+b
-	a-b
*	a*b
/	a/b
%	a%b

Control Statements


Control Statements
Decision Making
Loops
if
else if
switch
for
while
do-while

Decision Making

- one or more conditions to be evaluated or tested by the program
- statement or statements to be executed if the condition is determined to be true
- other statements to be executed if the condition is determined to be false

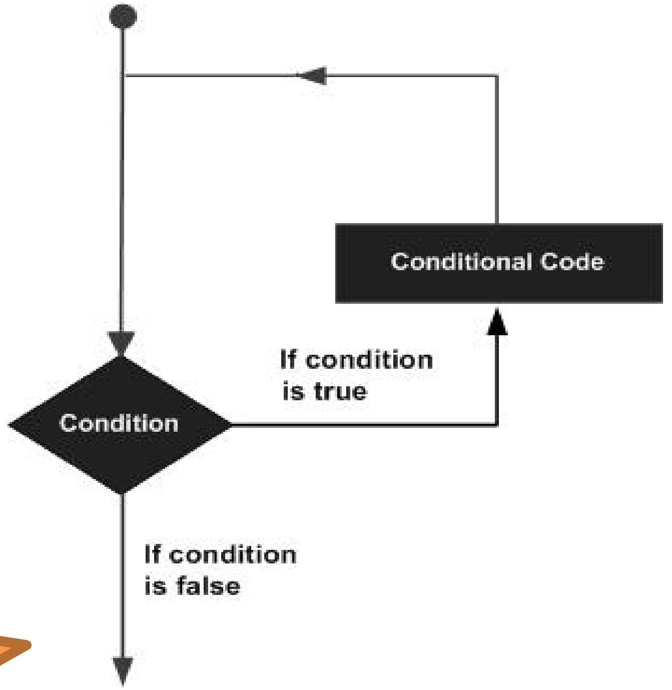


if
else if
switch

Loops

- block of code needs to be executed several number of times
- statements are executed sequentially

for
while
do-while



Decision Making

if

```
if(expression)
{
  //code to be executed
}
```

```
if(expression)
{
   //code to be executed
}
else
{
    //code to be executed if condition is false
}
```

- check some given condition and perform some operations depending upon the correctness of that condition

Example

Ex. Even odd number

```
int num=10;

if(num%2==0)
{
   printf("Even Number");
}
else
{
  printf("Odd Number");
}
```

Decision Making

else if

FortuneCloud
We Understand Technology

```
if(condition1)
{
     //code to be executed if condition1 is true
}
else if(condition2)
{
    //code to be executed if condition2 is true
}
else if(condition3)
{
    //code to be executed if condition3 is true
}
else
{
 //code to be executed if all the conditions are false
}
```

Multiple cases to be performed for different conditions

Example

Ex. Display days name

```
int day=1;

if(day==1)
{
   printf("Sunday");
}
else if(day==2)
{
  printf("Monday");
}
…….
else
{
  printf("Day not found");
}
```

Decision Making

switch

```
switch(expression)
{
    case value1:
         //code to be executed;
         break;

    case value2:
        //code to be executed;
        break;
    ......

    default:
        code to be executed if all cases are not matched;
}
```

Alternate to if-else-if statement which allows us to execute multiple operations

Example

```
int choice=1;

switch(choice)
{
   case 1:
     printf("Case 1 executed");
     break;

   case 2:
     printf("Case 2 executed");
     break;

   default:
      printf("Wrong choice");
}
```

Loops → for

```
for(initialization;condition;incr/decr)
{
    //code to be executed
}
```

iterate the statements or a part of the program several times
used to traverse the data structures like the array and linked list.

Example

Ex. Display 1 to 10 numbers

```
int i;

for(i=1;i<=10;i++)
{
   printf(“%d”,i);
}
```

Loops

while

```
while(condition)
{
   //code to be executed
}
```

Mostly used in the case where the number of iterations is not known in advance.

Example

Ex. Display 1 to 10 numbers

```
int i=1;

while(i<=10)
{
   printf("%d",i);
   i++;
}
```

Loops → do-while

```
do
{
   //code to be executed
} while(condition);
```

- post tested loop
- repeat the execution of several parts of the statements
- termination condition depends upon the end user.

Example

Ex. Display 1 to 10 numbers

```
int i=1;

do
{
   printf("%d",i);
   i++;
} while(i<=10);
```

Difference between while and do-while

- While loop is executed only when given condition is true.
- Condition is checked first then statement(s) is executed.
- while loop is entry controlled loop.

- do-while loop is executed for first time irrespective of the condition. After executing while loop for first time, then condition is checked.
- Statement(s) is executed atleast once, thereafter condition is checked.
- do-while loop is exit controlled loop.

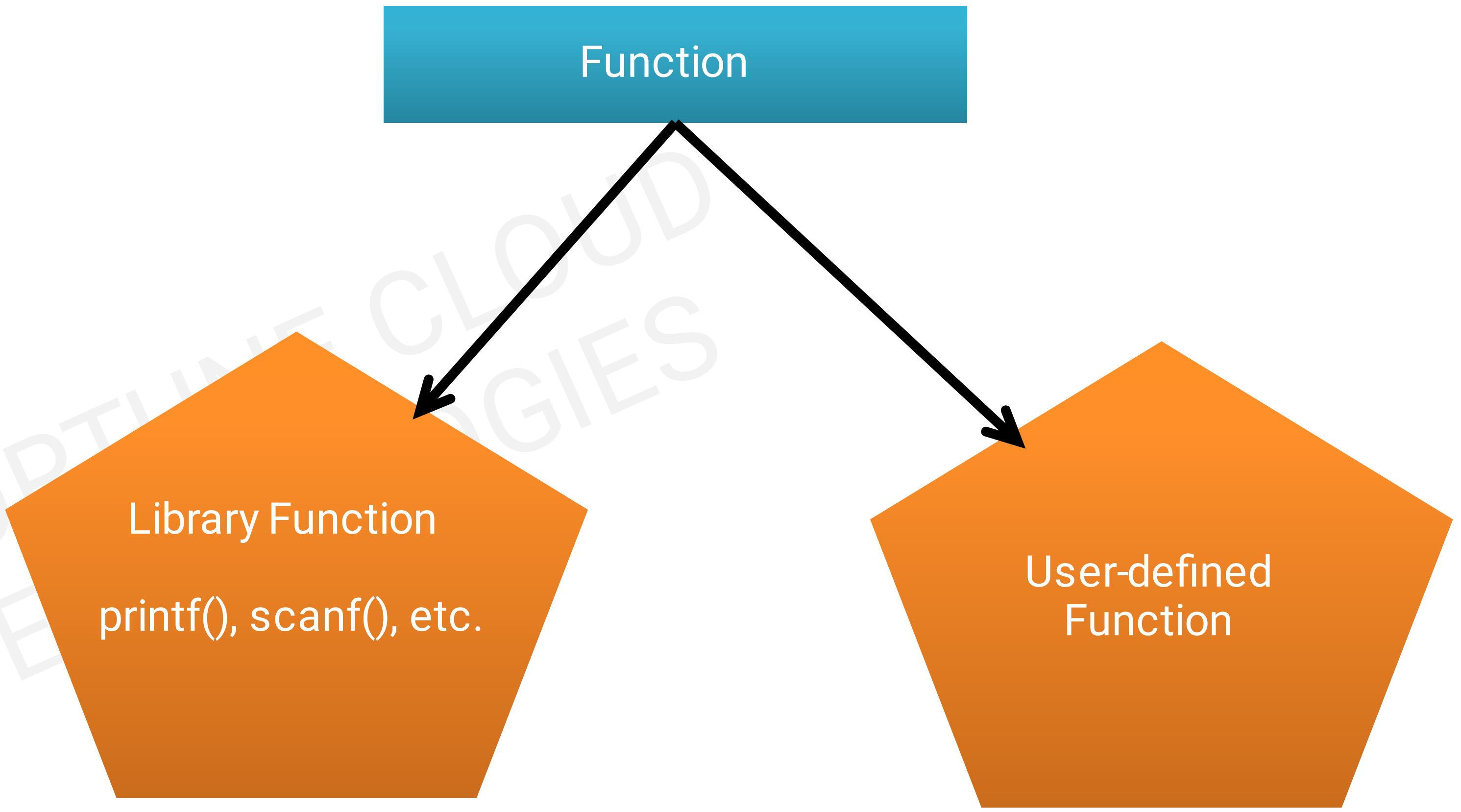
Function

- we can divide a large program into the basic building blocks known as function.
- It contains the set of programming statements enclosed by {}.

Advantages

- we can avoid rewriting same logic/code again and again.
- We can call C functions any number of times in a program.
- We can track a large C program easily when it is divided into multiple functions.
- Reusability is the main achievement.

Types of Functions



User-defined function

Default Function

```
void display()
{
    printf(“default function”);
}

void main()
{
    display();
}
```

User-defined function

Parameterized Function

```
int c; //global variable
void addition(int a, int b)
{
    c=a+b;
    printf("addition=%d",c);
}

void main()
{
int no1,no2; //local variable
printf("Enter two nos:");
scanf("%d%d",&no1,&no2);
addition(no1,no2);
}
```

User-defined function

Function with return

```
int addition(int a, int b)
{
    int c;
    c=a+b;
    return c;
}

void main()
{
    printf("%d",addition(10,20));
}
```

Use of Static Variable in function

```
void display()
{
    int a=10;   //local variable
    static int b=10;    //static variable

    a=a+1;
    b=b+1;
    printf("\n%d,%d",a,b);
}

void main()
{
            display();
            display();
            display();
}
```

Declared with the static keyword

Output:

```
11,11
11,12
11,13
```

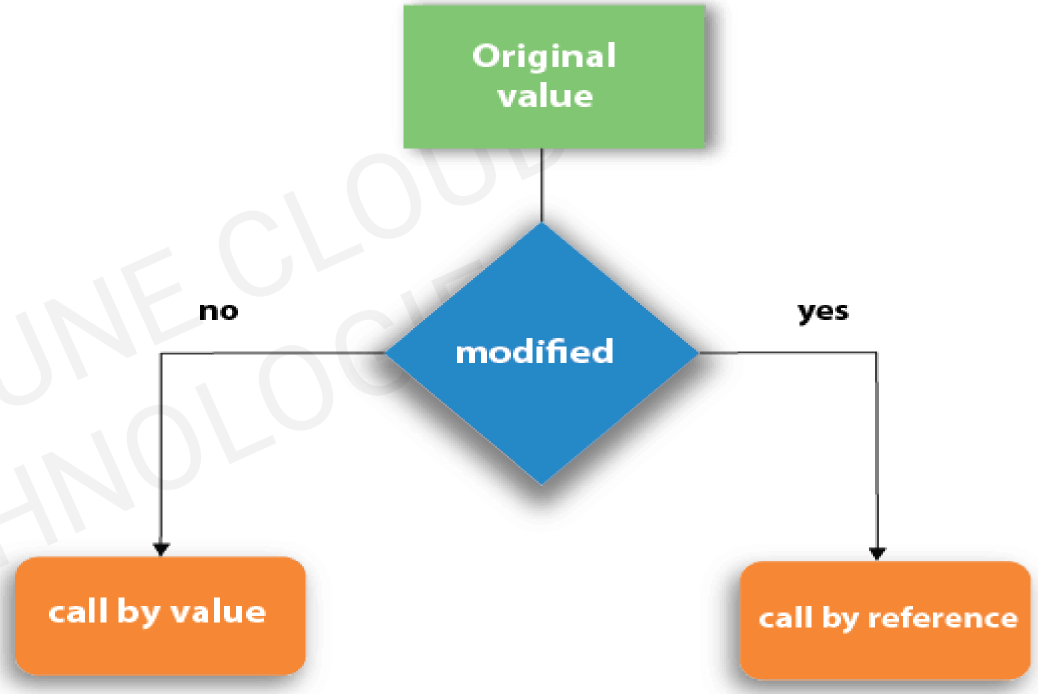
Recursive function

A recursive function is a function that calls itself and controlled by condition.

```
#include<stdio.h>
long int fact(int n);
int main()
 {
   int n;
   printf("Enter a positive integer: ");
   scanf("%d",&n);
   printf("Factorial of %d = %ld", n, fact(n));
   return 0;
}
```

```
long int fact(int n) {
   if (n>=1)
      return n*fact(n-1);
   else
      return 1;
}
```

Call by value and Call by reference



Call by value

```
void show(int num)
 {
   printf("\nBefore adding=%d",num);        // 10
   num=num+10;
   printf("\nAfter adding=%d", num);        // 20
}

int main()
{
   int a=10;
   printf("\nBefore function call a=%d", a);
                                               // 10
   show(a);

   printf("\nAfter function call a=%d", a);
   return 0;                                   // 10
}
```

- value of the actual parameters is copied into the formal parameters
- can not modify the value of the actual parameter
- different memory is allocated for actual and formal parameters

Call by reference

```
void show(int *num)
 {
   printf("\nBefore adding=%d",*num);      // 10
   (*num)+=10;
   printf("\nAfter adding=%d", *num);      // 20
}

int main()
{
   int a=10;
   printf("\nBefore function call a=%d", a);
                                              // 10
   show(&a);

   printf("\nAfter function call a=%d", a);
   return 0;
                                              // 20
}
```

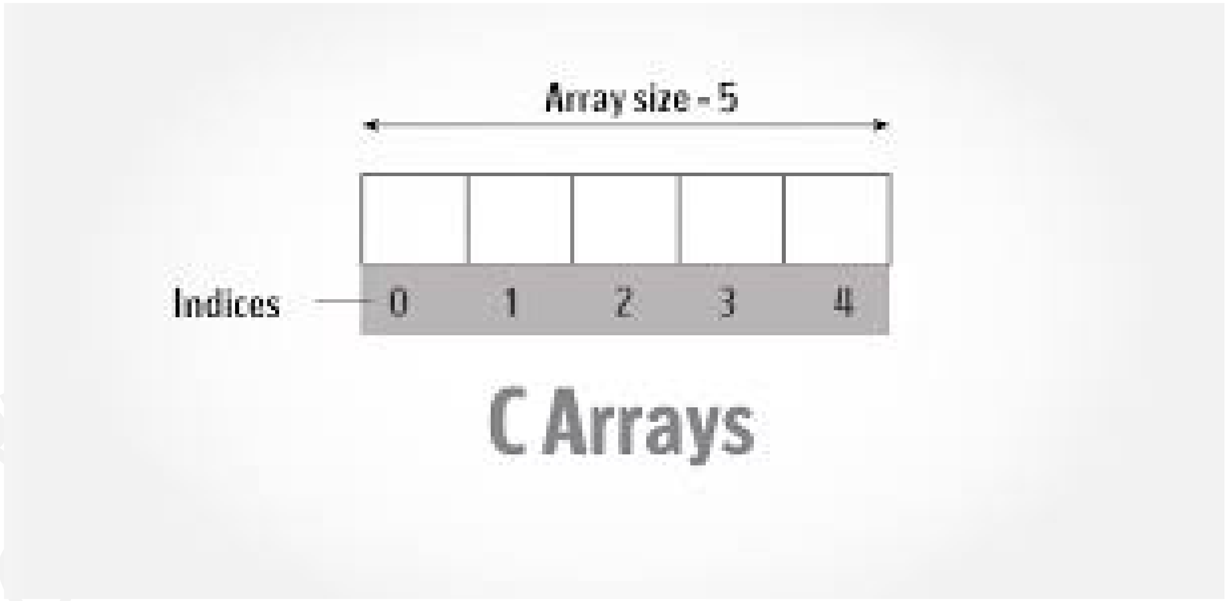
- the address of the variable is passed into the function call as the actual parameter.
- the memory allocation is similar for both formal parameters and actual parameters.
- modified value gets stored at the same address.

Arrays

- fixed-size sequential collection of elements of the same type.
- collection of similar type of data items stored at contiguous memory locations.
- store the primitive type of data such as int, char, double, float, etc.
- each data element can be randomly accessed by using its index number.
- All arrays consist of contiguous memory locations.
- The lowest address corresponds to the first element and the highest address to the last element.

Advantages

- Code Optimization
- Ease of traversing
- Ease of sorting
- Random Access


Array size - 5
Indices
0
1
2
3
4
C Arrays

Types of Array

One dimensional array

Two dimensional array

Multi-dimensional array

One dimensional array

```
int num[3];

num[0]=10;//initialization of array
num[1]=20;
num[2]=30;

printf("%d",num[0]);
printf("%d",num[1]);
printf("%d",num[2]);
```

OR

```
int i;
for(i=0;i<3;i++)
{
   printf("%d ",num[i]);
}
```

Two dimensional array

```
int num[2][2]={{1,2},{3,4}};


int i,j;
for(i=0;i<2;i++)
{
   for(j=0;j<2;j++)
   {
      printf("%d ",num[i][j]);
   }
}
```

Multi-dimensional array

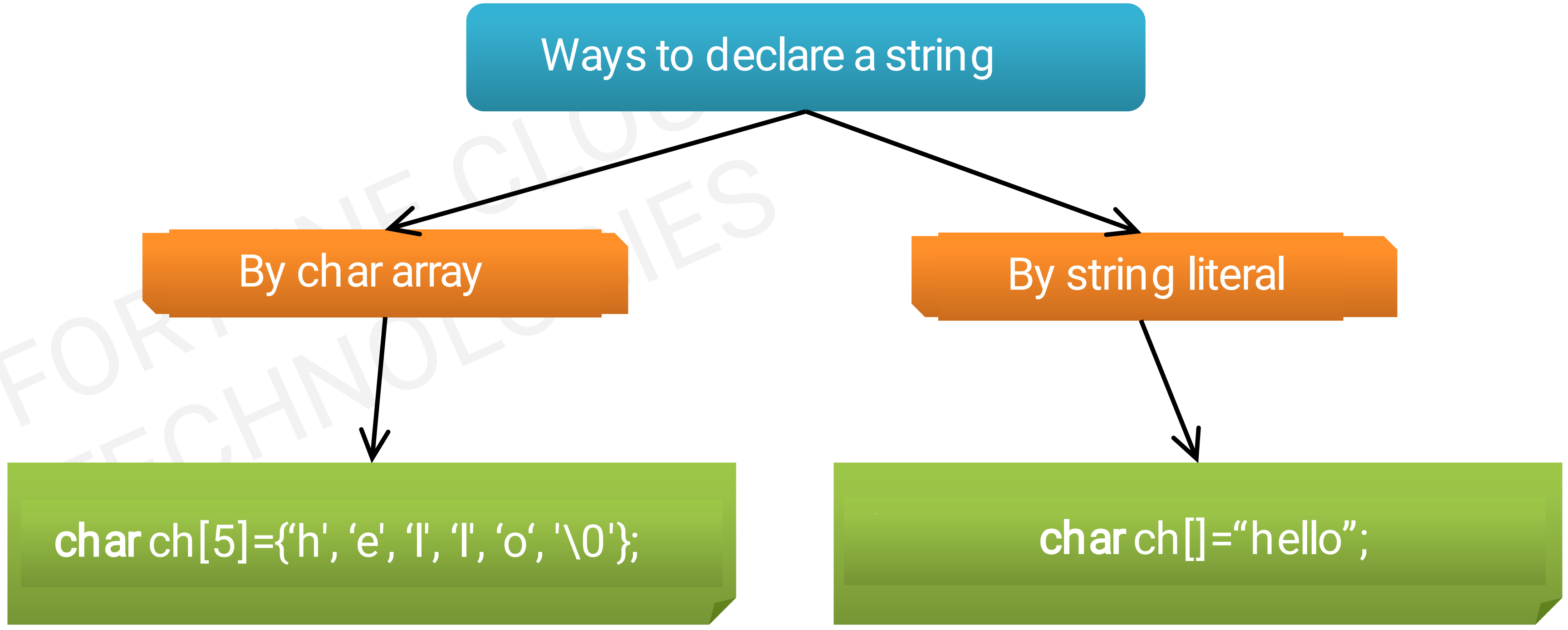
```
int num[2][2][2];


int i,j,k;

for(i=0;i<2;i++)
{
  for(j=0;j<2;j++)
  {
    for(k=0;k<2;k++)
    {
        printf("%d ",num[i][j][k]);
    }
  }
}
```

Example: Find out sum of elements in a given array

```
int num[5];
int i,sum=0;

printf("Enter the array elements");

for(i=0;i<5;i++)
{
   scanf("%d",&num[i]);
}

for(i=0;i<5;i++)
{
   sum=sum+num[i];
}

printf("Sum=%d",sum);
```

Strings

- one-dimensional array of characters terminated by a **null** character '\0'.
- The character array or the string is used to manipulate text such as word or sentences.



```
char ch[5]={‘h', ‘e', ‘l', ‘l', ‘o‘, '\0'};
```

```
char ch[]=“hello”;
```

gets() and puts()

```
char str[30];

printf("Enter the string");
scanf("%s",&str);

printf("String=%s",str);
```

Both the functions are involved in the input/output operations of the strings

```
char str[30];

printf("Enter the string");
gets(str);

printf("String=");
puts(str);
```

```
char ch1[10]="hello";
char ch2[10]="world
";
char ch3[10];
```

String Functions

strlen(ch1)	strlen	strrev	strrev(ch1)
strcpy(ch3,ch1)	strcpy	strlwr	strlwr(ch1)
strcat(ch1,ch2)	strcat	strupr	strupr(ch1)
strcmp(ch1, ch2)	strcmp	strstr	strstr(ch1,"substring")

Example: Count number of vowels in a string

```
char str[100];
int i=0, count=0;

printf("Enter a string\n");
gets(str);

while (str[i] != '\0')
 {
   if (str[i] == 'a' || str[i]  == 'A' || str[i]  == 'e' || str[i]  == 'E' || str[i]  =
= 'i' || str[i]  == 'I' || str[i]  =='o' || str[i] =='O' || str[i]  == 'u' || str[i]  ==
'U')
     count++;
   i++;
 }
 printf("Number of vowels in the string: %d", count);
 return 0;
```

FortuneCloud
We Understand Technology

Structure-Union

- Structure in c is a user-defined data type that enables us to store the collection of different data types.
- Each element of a structure is called a member.
- **struct** keyword is used to define the structure.

Why use structure

- There are cases where we need to store multiple attributes of an entity.
- It can have different attributes of different data types.

Declaring Structure

By struct keyword

```
struct employee
{  int id;
   char name[20];
   float salary;
};
```

```
struct employee e1,
e2;
```

By declaring a variable

```
struct employee
{  int id;
   char name[20];
   float salary;
}e1,e2;
```

Example:

```
struct employee
{   int id;
    char name[20];
}e1;

int main( )
{
   e1.id=101;
   strcpy(e1.name, “abc");

   printf( “Id : %d", e1.id);
   printf( “\nName : %s", e1.name);
return 0;
}
```

Array of structure

collection of multiple structures variables

```
struct student
{
    int id;
    char name[10];
};

int main()
{
  struct student s[5];
  int i;

for(i=0;i<5;i++){
   printf("\nEnter Id:");
   scanf("%d",&s[i].id);
   printf("\nEnter Name:");
   scanf("%s",&s[i].name);
}
```

```
printf("\nStudent Information\n");

for(i=0;i<5;i++){
printf("\nId:%d, Name:%s",s[i].id,s[i].name);
}
  return 0;
}
```

Union

- Store different data types in the same memory location
- It **occupies less memory** because it occupies the size of the largest member only.

```
union employee
{   int id;
    char name[20];
}e1;

void main()
{
   e1.id=101;
   strcpy(e1.name, “abcd");

   printf( "id : %d\n", e1.id);
   printf( "name : %s\n", e1.name);
}
```

id gets garbage value because name has large memory size

Difference Between Structure And Union

- Separate memory location is allotted to each input member
- **struct** student
```
{
  int id;
  char name[20];
};
```
- Size of Structure is equal or greater than the sum of size of all the data members.

- Memory is allocated only to one member having largest size among all other input variables
- **union** student
```
{
  int id;
  char name[20];
};
```
- Its size equal to the size of largest member among all data members.

Pointers

- A **pointer** is a variable whose value is the address of another variable, i.e., direct address of the memory location.
- The pointer in C language is a variable which stores the address of another variable.
- This variable can be of type int, char, array, function, or any other pointer.

Example

```
int num = 10;
int *p = &num;
```

Advantages

Pointer **reduces the code** and **improves the performance**

It makes you able to **access any memory location**

Address Of (&) Operator

```
int num=10;

printf("value of number is %d",num);

printf("address of number is %u",&num);
```

Returns the address of a variable

we need to use %u to display the address of a variable.

NULL Pointer

- A pointer that is not assigned any value but NULL is known as the NULL pointer.

```
int *ptr=NULL;

 if(ptr!=NULL)
 {
   printf("value of ptr is : %d",*ptr);
 }
 else
 {
   printf(“Null pointer");
 }
```

Pointer Arithmetic

```
int a=10,b=20,c;
int *p,*q;

p=&a;
q=&b;

c=*p+*q;

printf("addition: %d",c);
```

sizeof() operator

```
int a=10;


printf("%d",sizeof(a));

//sizeof(int);
//sizeof(char);
//sizeof(float);
```

It determines the size of the expression or the data type

Function pointer

int (*p) (**int** , **int**);   //Declaration of a function pointer.

int add(**int** , **int**);   // Declaration of function.

p = add; // Assigning address of add to the p pointer.

We can get the address of memory by using the function pointer.

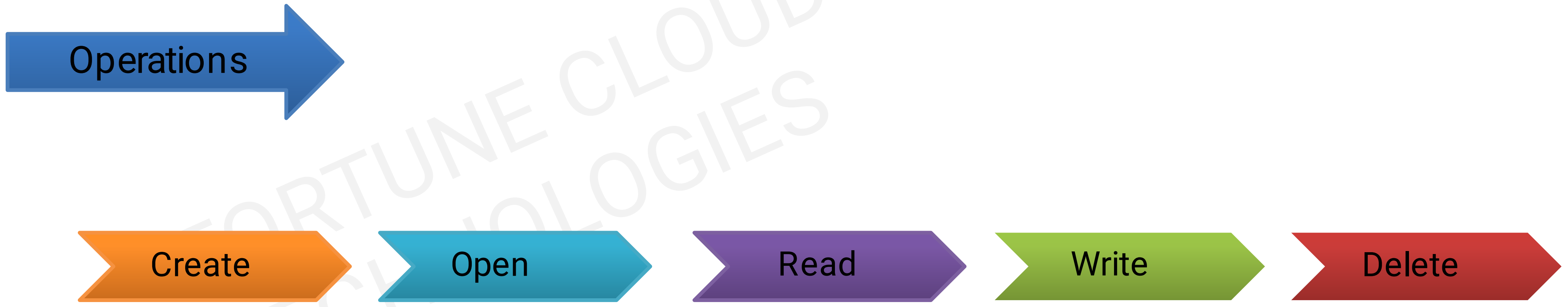
Example

```
int add(int a, int b)
{
   int c;
   c=a+b;
   return c;
}
```

```
int main()
{
  int a,b;

  int (*p)(int,int);

  int result;
  printf("Enter  a and b : ");
  scanf("%d%d",&a,&b);

  p=add;

  result=(*p)(a,b);

  printf("Addition: %d",result);

   return 0;
}
```

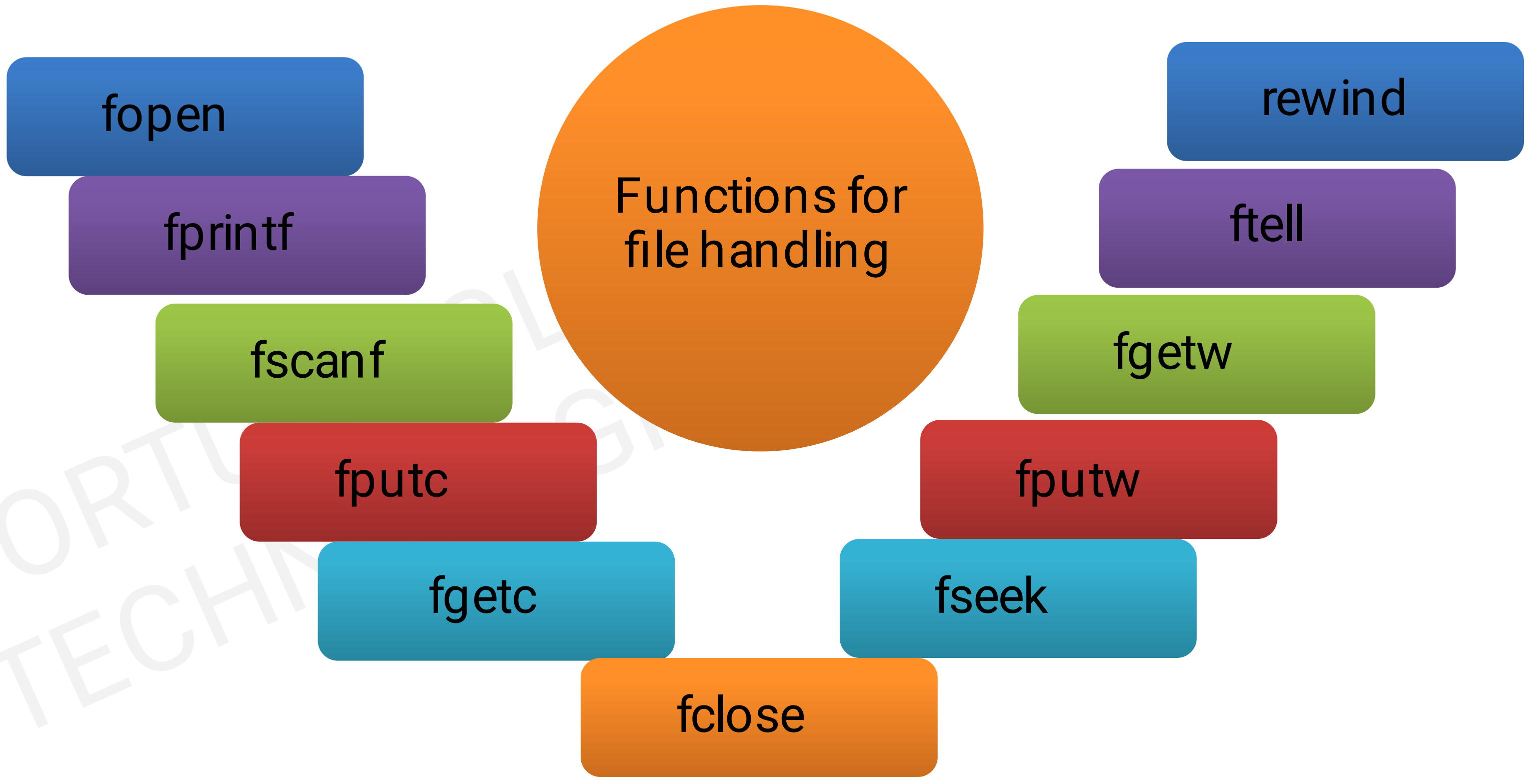
File Handling

- **File Handling** is the storing of data in a **file** using a program.
- File handling in C enables us to create, update, read, and delete the files stored on the local file system through our C program.



Modes

r	Opens an existing text file for reading purpose.
w	Opens a text file for writing. If it does not exist, then a new file is created.
a	Opens a text file for writing in appending mode. If it does not exist, then a new file is created.
r+	Opens a text file for both reading and writing.
W+	Opens a text file for both reading and writing. It first truncates the file to zero length if it exists, otherwise creates a file if it does not exist.
a+	Opens a text file for both reading and writing. It creates the file if it does not exist. The reading will start from the beginning but writing can only be appended.


Functions for file handling
fopen
fprintf
fscanf
fputc
fgetc
fclose
fseek
fputw
fgetw
ftell
rewind

fprintf() and fscanf()

```
FILE *fp;
fp = fopen("file.txt", "w");//opening file

fprintf(fp, "Hello file by fprintf...\n");//writing data into file

fclose(fp);//closing file
```

used to write set of characters into file

used to read set of characters from file

```
FILE *fp;
char buff[255];//creating char array to store data of file
fp = fopen("file.txt", "r");

while(fscanf(fp, "%s", buff)!=EOF)
{
  printf("%s ", buff );
}
fclose(fp);
```

fputc() and fgetc()

```
FILE *fp;
fp = fopen("file.txt", "w");//opening file

fputc('a',fp);//writing single character into file

fclose(fp);//closing file
```

used to write a single character into file

returns a single character from the file

```
FILE *fp;
char c;

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

while((c=fgetc(fp))!=EOF)
{
   printf("%c",c);
}
fclose(fp);
```

fputs() and fgets()

```
FILE *fp;

fp=fopen("myfile.txt","w");
fputs("hello c programming",fp);

fclose(fp);
```

writes a line of characters into file

reads a line of characters from file

```
FILE *fp;
char text[300];

fp=fopen("myfile.txt","r");
printf("%s",fgets(text,200,fp));

fclose(fp);
```

fseek()

```
FILE *fp;

fp = fopen("myfile.txt","w+");

fputs(“C programming", fp);

fseek( fp, 7, SEEK_SET );

fputs(“Practical", fp);

fclose(fp);
```

It is used to set the file pointer to the specified offset. It is used to write data into file at desired location.

rewind()

- sets the file pointer at the beginning of the stream
- It is useful if you have to use stream many times.

```
FILE *fp;
char c;

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

while((c=fgetc(fp))!=EOF){
printf("%c",c);
}

rewind(fp);

while((c=fgetc(fp))!=EOF){
printf("%c",c);
}

fclose(fp);
```

ftell()

```
FILE *fp;
int length;

fp = fopen("file.txt", "r");
fseek(fp, 0, SEEK_END);

length = ftell(fp);

fclose(fp);

printf("Size of file: %d bytes", length);
```

- returns the current file position of the specified stream
- get the total size of a file after moving file pointer at the end of file.

Thank you !