C – LANGUAGE

INDEX

- Global scope (global variable)
- Local scope (local variable)

12. Pointers
    - Address operator, value operator
    - Pointer to pointer
13. Pass by value and pass by reference
14. Preprocessor directives.
    - Introduction
    - Function like macros
    - Conditional compilation directives
15. File handling
    - Creating a file
    - Writing in the file
    - Reading from the file
    - Renaming a file
    - Removing a file.
16. Predefined Functions
    - Strcpy(),strcat(),strlen(),stcmp(),strlwr(),strupr(),

01. CONSTANTS, VARIABLES, DATA TYPES, KEYWORDS

CONSTANTS

- A constant is an entity that doesn't change.
- Ex:
  - ✓ Integer constants : 2, 50, 9999
  - ✓ Real constants : 25.000, 51.11478, 198.9169
  - ✓ Character constants: 'a', 'A', '?', '9'

VARIABLES

- A Variable is an entity that change
- In programming we have to store the result of calculations, and it is stored in the computer memory
- MEMORY ALLOCATION

X

768963

10		

- **Computer memory contains lots of memory cells, with different addresses, here 768763. Which store a constant value 10. We name that memory cell as X.**
- **Variable names are the names which are given to the memory cells.**

TYPES OF VARIABLES (DATA TYPES):

- **The memory cells can contain integer, floating point and character constants**
- **Type of variable is nothing but the type of constant that the variable can store**

VARIABLE TYPE	KEYWORD	EXAMPLE
Integer	Int	Int a; Int age;
Real	Float	Float x, Float average;
Character	Char	Char s; Char inti;

KEYWORDS:

- **By using keywords we can construct meaningful statements**
- **These are the words whose meaning are predefined in the c compiler**
  Eg: break, int, float, if, for etc.,
- **Key words can't be used for variable names.**

02. HELLO WORLD

```
#include<stdio.h>
#include<conio.h>
Int main()
{
Printf("hello world");
Getch();
Return 0;
}
```

Note: stdio(standard input and output) are header files(.h). which are included throw preprocessor directive. A preprocessor is a software which processes our source code before the compilation.

03. VARIABLES

```
#include<stdio.h>
#include<conio.h>
int main()
{
int x;
float y;
char z;

x=10;
y=20.00;
z='k';

printf("%d %f %c",x,y,z);

getch();
return 0;
}
```

04. PRINTF (), SCANF ()

a. escape sequence:

\n : next line
\t : one tab space
\ : printf("my name is \"kiran\" my age is %d", x);

b. scanf():

```
#include<stdio.h>
#include<conio.h>
int main()
{
int age;
int weight;
printf("enter you age and weight");
scanf("%d %d",&age,&weight);
printf("my age is %d \n my weigh is %d",age,weight);
getch();
}
```

05. CONDITIONAL STATEMENTS

1.if

```
#include<stdio.h>
#include<conio.h>
int main()
{
int age;
int weight;
age=22;
if (age>10 and age<20)
{
printf("adult");
}
```

2.if else

```
if(age>20 and age<40)
{
        printf("old");
}
else
{
        printf("young");
}
getch();
}
```

3. Ternary operator:

Synt:

Condition ? statement1 : statement2;

If the given condition is true then stat1 is executed, if the given condition is false statement2 is executed.

Ex1:

```
#include<stdio.h>
#include<conio.h>
int main()
{
int age=10;
```

```
age>=18? printf("adult") : printf("teen");
getch ();
return 0;
}
```

Ex2:

```
#include<stdio.h>
#include<conio.h>
int main()
{
int age=10;
int x;
x = (age>=10) ? 1 : 0 ;
printf("%d",x);
getch();
return 0;
}
```

4.switch

we use switch statements where we have multiple conditions

synt:

```
switch (expression)
{
        Case constant1;
        Statements;
        Break;
        Case constant2;
        Statements;
        Break;
        :
        :
        Default:
        Statements;
}
```

Ex:

```
int main()
	{
		char age='A';
		switch (age)
			{
			case 'a':
				printf("teen");
				break;
			case 'A':
				printf("adult");
				break;
			}
	getch();
	return 0;
	}
```

Note: **In switch we cant use float for ex switch(x){ case 1.1}**

06.OPETATORS

a.arithmetic operators

+,
-,
*** ,**
/, **(10/2, 2/10)**
mod() **‘%’** **—mod returns give remainder,**

++ (increment)
Int n=10
N++5;
Printf(“%d”,n);
o/p=5;

--(decrement)

b. logical operator:

AND &&
OR ||

```
Int age;
Age=20;

If (age>10 && age<20)
{
Printf(“teen”)
}
Else
{
Printf(“adult”);
}
```

C. RELATIONAL OPERATOR:

>, <, <=, >=, =, !=

D. INCREMENT(++) AND DECREMET(--) OPERATOR:

- **These are also called short hand operators**
- **When we use this operator in suffix its meaning is different comparing to usage in prefix**
- **X++ (suffix): increment occur after execution of that statement.**
- **++X (prefix): increment is done first, then it is assigned to variable.(in the same statment)**
- **X--, --X are similar to that**

Ex:

```
#include<stdio.h>
#include<conio.h>
int main()
{
int x=10;
printf("x value is %d\n",x++);
printf("x value is %d\n",x);
```

```
int y=100;
printf("y value is %d\n",++y);
printf("y value is %d\n",y);

getch();
return 0;

}
```

o/p:

```
x value is 10
x value is 11
y value is 101
y value is 101
```

07.LOOPING

1.While loop:

Synt:

```
While (conditions)
{
Statements;
}
```

Ex:

```
int main()
{
        int a=1;
        while(a<=10)
        {
        printf("%d",a);
        a++;
        }
        getch();
        return 0;
}
```

2. do while loop

Synt:

In while loop first condition is checked and then statements in the loop are executed, where as in do while loop first statement is executed and then condition is checked.

If condition is failed in while loop then no statement is executed
If condition is failed in do while loop then one statement is executed. i.e., in do while loop atleast one statement is executed.

Dowhile loop is mainly used in mainly in menu selection function.

Synt:

```
do
{
Statements;
}while(condition);
```

Ex:

```
int main()
{
        int x=0;
        do
        {
        printf("%d\n",x);
        x++;
        }while(x<=5);
        getch();
        return 0;
}
```

3.for loop:

Synt:

```
For (initialize counter; condition; increment or decrement)
{
Statements;
}
```

Ex:

```
int main()
{
        int x;
        for(x=0;x<=10;x++)
        {
        printf("%d\n",x);
        }
        getch();
        return 0;
}
```

Ex2: (two counters at a time)

We cant use two conditions in for loop like below

For (x=10, y=20; x<100,Y<=200 ; x++, y--)

Instead we can use logical operator in condition like below

For (x=10, y=20; x<100||Y<=200 ; x++, y--)

Ex3: (infinite for loop)

```
Int main
{
        Int x;
        For( ; ;)
        {
        Printf("%d\n",x)
        X++;
        If(x==5)
        Break;
        }
}
```

08.LOOP BREAKERS

1.break:

- **"Break" statement will stop the looping and exit from the loop**

- **If we are in nested loop (loop with in loop), If we use break in inner loop then we exit from inner loop**
- **Exit from switch loop.**

Ex:

```
int main()
{
        int x;
        for (x=0;x<=5;x++)
        {
        printf("%d",x);
        if (x==3)
        break;
        }
        getch();
        return 0;
}
```

2.continue:

- **“continue ” will help to skip that single loop.**
- **i.e, after executing continue statement it will again go back to starting of loop**

Ex:

```
int main()
{
        int x;
        for (x=0;x<=5;x++)
        {
        if (x==3)
        continue;
        printf("%d",x);
        }
        getch();
        return 0;
}
```

o/p:

01245

Here 3 is not printed.

09.ARRAYS

- **if we create one variable, we can only store one value in it,**
- **for ex if we want to store 1000 student marks then we have to create 1000 variables which is not advisable. So the concept of arrays is introduced,**
- **by creating single array we can store multiple values in it.**
- **It can only store only one kind of variables**
- **Size of the array is not mandatory,**

1.integer arrays:

Ex1:

```
int main()
	{
		int marks[6]={6,7,8,9,20,19};
		printf("%d",marks[0]);
		printf("%d",marks[2]);
		printf("%d",marks[5]);
	getch();
	return 0;
	}
```

Ex2:

```
int main()
	{
		int i=3;
		int marks[i];
		printf("enter the marks\n");

		for (i=0;i<=2;i++)
		{
		scanf("%d",&marks[i]);
		}

		for (i=0;i<=2;i++)
```

```
        {
        printf("marks of element %d is %d\n",i,marks[i]);
        }

    getch();
    return 0;
    }
```

2.character or string arrays:

In c string a combination of characters and on last one it is a null value.
So while printing it print some random value to fill that null valve.
To over come this we add the null value to characters

```
int main()
    {
        char x[ ]={ 'a','b','c','d'};
        printf("%c\n",x[0]);
        printf("%s\n",x);

        char x1[ ]={'a','n','i','l','\0'};
        printf("%s\n",x1);

        char x2[ ]="kiran";
        printf(" %s\n",x2);
        printf("%c\n",x2[3]);

    getch();
    return 0;
    }
```

10.FUNCTIONS

1.introduction:

a function is a collection of statements, which is going to do a particular task.
Synt:
Return_type function_name (parameters)
{

Function body;
}

Ex:

```
void display();        --- function protype (we are informing compiler that this function is created some where else )
int main()
{
display();
getch();
return 0;
}

void display()
{
printf("hello world");
}
```

Note: if don't use function prototype, then compiler comes at the function calling statement(here display();). Then it gives error saying that function is not defined.

Sol:

```
void display()
{
printf("hello world");
}

int main()
{
display();
getch();
return 0;
}
```

If you don't want to use the function prototype, then you have to declare that function before calling it.

2.control transfer.

```
void function1();
```

```
void function2();

int main(){
printf("in main function\n");
printf("calling function1\n");
function1();
printf("in main funciton after retruning from funciton1\n");
printf("finishing main function\n");

getch();
return 0;
}

void function1(){
        printf("in function1\n");
        printf("calling function2\n");
        function2();
        printf("in function1 after returning from function2\n");
        printf("returning from function1\n");
}

void function2(){
        printf("in function2\n");
        printf("returning from function2\n");
        }
```

3.parameters

```
void add(int, int);

int main(){
add(2,3);
getch();
return 0;
}

void add(int a, int b){
```

```
        int c;
        c=a+b;
        printf("add is %d",c);
}
```

4.return statement:

```
int add(int, int);

    int main(){
    int result;
    result = add(2,3);
    printf("the result is %d\n",result);
    getch();
    return 0;
    }

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

5.passing whole array to function:

```
    void display(int);

    int main(){
    int marks[ ]={10,20,30};
    int i;
    for (i=0; i<=2; i++){
    display(marks[i]);
    }
    getch();
    return 0;
    }

    void display(int mark){
        printf("the marks received is %d\n",mark);
```

```
}
```

(OR)

```
void display(int, int);

    int main(){
    int marks[]={10,20,30};
    display(marks,3);
    getch();
    return 0;
    }

    void display(int mark[ ], int s){
        int c;
        for(c=0; c<s; c++){
        printf("the array elemet at  %d is %d\n",c,mark[c]);
        }
    }
```

11.VARIABLE SCOPE

1. global variable:

It is available in all functions.

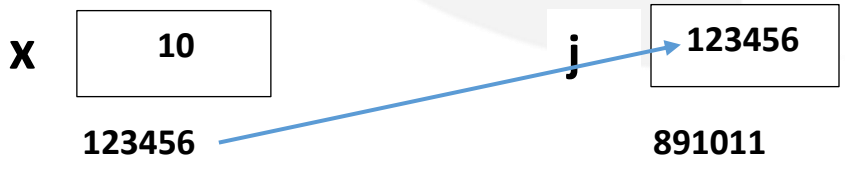
2. local variable:

it is available in only one function where it is declared.

12.POINTERS

1.introduction:

Int x=10; **int *j;**



Note: %p

& address of operator: when used before a variable its return the physical address of that variable
***- value of operator: it should be used before the address value, then it returns the value stored in that address.**

Ex:

```
int main(){
        int x=10;
        int *j;
        j = &x;  --- pointer

        printf("the value of x is%d\n",x);
        printf("the address of  x is %p\n",&x);
        printf("the value of *(&x) is %d\n\n", *(&x));

        printf("the address value in j is %p\n",j);
        printf("the address of j is %p\n",&j);
        printf("the value of x is %d\n",*j);
        printf("the value of *(&j) is %p\n",*(&j));

        getch();
        return 0;
}
```

2.pointer to pointer:

```
int main(){
        int x=10;
        int *p;
        int **q;

        p=&x;
        q=&p;

        printf("the value of  x is %d\n",x);
        printf("the address of x is %p\n\n",&x);

        printf("the value in p is %p\n",p);
```

```
        printf("the address of p is %p\n",&p);
        printf("the value stored in p is %d\n",*p);

        printf("the value in q is %p\n",q);
        printf("the address of q is %p\n",&q);
        printf("the value stored in q from p from x is %d\n",**q);

        getch();
        return 0;

}
```

13.pass by value and pass by reference

1.pass by value

```
void display(int, int) ;
int main(){
        int x=10;
        int y=20;
        display(x,y);
        getch();
        return 0;
}

void display(int a, int b){
        printf("a is %d b is %d\n",a,b);
        a=100;
        b=200;
        printf("a is %d b is %d\n",a,b);
}
```

2.pass by reference:

```
void display(int *, int *) ;
int main(){
        int x=10;
        int y=20;

        display(&x,&y);
```

```
        getch();
        return 0;
}

void display(int *p, int *q){
        printf("a is %d b is %d\n",*p,*q);

}
```

14.preprocessor directives

1.introduction:

A preprocessor is a program which gonna process our source code, before it is passed to the compiler, for the compilation process, so this preprocessor offers several features called preprocessor directives

#define preprocessor directive:

Synt:

#define macro_name character_sequence (to end pre processor directive we press enter)

Ex:

```
#include<stdio.h>
#include<conio.h>
#define LIMIT 10
int main(){
        int c;
        for (c=0; c<=LIMIT;c++){
                printf("%d\n",c);
        }
        getch();
        return 0;
}
```

EX2:

```
#include<stdio.h>
#include<conio.h>
#define  LIMIT  10
```

```
#define  TOP  LIMIT+1
int main(){
        int c;
        for (c=0; c<=TOP;c++){
                printf("%d\n",c);
        }
        getch();
        return 0;
}
```

Note: vairables can be changed in source code, but macros cant be changed

2.functions like macro:

Here macro_name or macro_temp will have macro_definition

Ex:

```
#define  CHECK(number) if(number>=10){\
                                        printf("the numenr is greater than 10");\
                                        }else{ \
                                        printf("the no is less than 10");\
                                        }\
int main(){

        CHECK(25);
        getch();
        return 0;
}
```

3.#include preprocessor directive:

Synt:

#include **<file_name>--** *it searches the file in default path()*

#include **"file_name"—***it searches in the same folder where the current file is saved if it not found then it going to search in default path.*

Ex:

Step1*:create a function and save it in as .c file in some folder*

```
void display(){
                printf("displayed from function\n");
```

```
}
```

Step2: *use that .c file in #include preprocessor directive*

```
#include "display.c"
int main()
{
        display();
        getch();
        return 0;
}
```

3.conditonal compilation directives:

#if #elif #else #endif

These are the directive provided by preprocessor in c. these allow us to selectively compile the portions of program source code

Ex:

```
#define MARK 90
 int main()
{
        #if MARK>=75
        printf("grade A\n");

        #elif MARK>=50
        printf("grade B\n");

        #else
        printf("grade C\n");

        #endif

        getch();
        return 0;
}
```

#ifdef #ifndef #undef #endif

```
#define INTEL

#define MSFT
#undef      MSFT

int main()
{
            #ifdef INTEL
            printf("this is intel pc\n");
            #endif

            #ifndef      QUALCOMM
            printf("this is not qualcomm pc\n");
            #endif

            #ifdef MSFT
            printf("this is MSFT OS\n");
            #endif

            #ifndef      MSFT
            printf("this is not MSFT OS\n");

            #endif

            getch();
            return 0;
}
```

15.FILE HANDLING

1.introdction:

- **File are the building blocks of every operating system.**
- **Files are the virtual container to store some virtual container of the data**
- **Generally we use variables to store data, but when program is terminated the data is lost. So, to store data permanently in memory we use files.**

How to create a file?

```
int main()
{
	FILE *fp;
	fp=fopen("kiran.txt","w");

	if(fp==NULL)
	{
		printf("unable to create file\n");
	}

	else
	{
		printf("file created successfull\n");
	}
	fFclose(fp);
	getch();
}
```

2.how to write to a file?

I method:

```
#include<cstring>
int main()
{
	char data[25]="roacking star";
	int length = strlen(data);
	int c;

	FILE *fp;
	fp=fopen("kiran.txt","w");

	if(fp==NULL)
	{
		printf("unable to create file\n");
	}

	else
	{
```

```
        for(c=0; c<length; c++)
        {
            printf("%c is entered to file\n",data[c]);
            fputc(data[c],fp);
        }
        printf("file created successfull\n");
    }

    getch();
    return 0;
}
```

II method: (directly)

```
int main()
{
    char data[25];

    FILE *fp;
    fp=fopen("kian.txt","w");

    if(fp==NULL)
    {
        printf("unable to create file");
    }

    else
    {
        printf("enter the data\n");
        gets(data);
        fputs(data,fp);
        printf("created successfully\n");
    }
    fclose(fp);

    getch();
    return 0;
}
```

III method: (formatted way)

```
int main()
{
	char data[25];
	int age;

	FILE *fp;
	fp=fopen("kian.txt","w");

	if(fp==NULL)
	{
		printf("unable to create file");
	}

	else
	{
		printf("enter the ur name and age");
		scanf("%s%d",data,&age);
		fprintf(fp,"%s\t%d",data,age);
		printf("created successfully\n");

	}
	fclose(fp);

	getch();
	return 0;
}
```

3.how to read from file?

I method:

```
int main()
{
	char data;

	FILE *fp;
```

```
	fp=fopen("kian.txt","r");

	if(fp==NULL)
	{
		printf("unable to create file");
	}

	else
	{
		while(!feof(fp))
		{
			data=fgetc(fp);
			printf("%c",data);
		}
	}
	fclose(fp);

	getch();
	return 0;
}
```

Note: feof: *returns '1' at the end of file*
Returns '0' for all other

II method:

```
int main()
{
	char data[25];

	FILE *fp;
	fp=fopen("kian.txt","r");

	if(fp==NULL)
	{
		printf("unable to create file");
	}
```

```
	else
	{
		while(!feof(fp))
		{
		fgets(data,25,fp);
		printf("%s",data);
		}
	}
	fclose(fp);

	getch();
	return 0;
}
```

III method:

```
int main()
{
	char data[25];
	int age;

	FILE *fp;
	fp=fopen("kian.txt","r");

	if(fp==NULL)
	{
		printf("unable to create file");
	}


	else
	{
		while(!feof(fp))
		{
		fscanf(fp,"%s%d",data,&age);
		printf("%s\t%d\n",data,age);
		}
	}
```

```
        fclose(fp);

        getch();
        return 0;
}
```

Getc(variable)	Putc(variable)
Gets(vara)	Puts(vat)
Scanf("%d",var)	Printf("%d",var)
Data=fgetc(fp)	Fputc(data,fp)
Fgets(data,25,fp)	Fputs(data[25],fp)
Fscanf(fp,"%d",var)-→foramtted way	Fprintf(fp,"%d",var)→formatted way

4.renaming a file:

Int rename("oldfile.txt","newfile.txt");
Returns '0' if renamed
Returns "non zero" if error in renaming

```
int main()
{
        if( rename("anil.txt","kin.txt")== 0 )
        {
                puts("successfull");
        }
        else
        {
                puts("not successfull");
        }

}
```

5.remove a file:

Int remove("filename.txt");
Return '0' if deleted;
Returns "non zero value" if error in deleting

```
int main()
{
	if( remove("kin.txt")== 0 )
	{
		puts("successfull");
	}
	else
	{
		puts("not successfull");
	}
}
```

16.PREDEFINED FUNCTIONS

1.strcpy()

```
int main()
{
	char source[]="anil shetty jas;kjfa;";
	char target[25];

	strcpy(target,source);
	printf("%s",target);

	getch();
	return 0;
}
```

2.strcat()

```
int main()
{
	char source[]="age is 21";
	char target[]="kiran ";

	strcat(target,source);
	printf("%s",target);
```

```
        getch();
        return 0;
}
```

3.strlen()

```
int main()
{
        char source[]="age is 21";

        printf("%d",strlen(source));

        getch();
        return 0;
}
```