C - PROGRAMING

—x—x—x—x—

program :- set of Instruction.
↓
set of data with keybord command
↓
raw meterial
↓
{a, 9, b, +, -, anything}

Computer language

↳ understands machine language
like :- Binary code (0, 1)

Note :-

How to work

Alphabet (A-Z, a-z)
Digit (0-9)
Special symbol (+, ", etc)
→ keyword, variable, Data type → Instruction
↓
program
↓
project or (S/W)

C programing language

High level language (Human readable) (english)
↓
Translate ~~throu~~ through compiler
↓
machine code (output)

→ Developed by Denis Ritchi in 1972 at AT & T (American Telephone & Telegraph) Bell laboratory in California university in USA.

→ Case sensitive language (name ≠ Name).

→ 32 keywords.

→ All keywords are reserve word.

→ All keywords are written in lower case.

→ middle level language. — [Low level language. / High level language.]

→ Mother language of all programing.

→ C is a structure programing language.

C - Tokens

→ A smallest individual units of a program is called Tokens. (symbol)

eg:-
```
a=10
b=6
c= 10+6
print c;
```

→ There are six types of Tokens in C.

C - Tokens

(1) keyword
(2) Identifier
(3) constants
(4) String
(5) special symbol
(6) operators

Note :-
= → Assignment operator
== → equals to

Keywords :-

ere are 32 keywords in C.
All keywords are written in lower se.
All keywords are used for specific sk.
Keywords can not be variable name.
All keywords are reserve words.

example :-

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

Identifier :-

n identifier is a variable name which used to allocate the memory location. also refer to the name of function Array.
Both uppercase & lowercase letters are permitted.

→ underscore are also permitted in identifier. which is used to link between two words in long identifier.

eg:- Student - Name , Sr-no
Student - Roll , Random-function

Rules for Identifier

→ first character must be an alphabets.
→ it must consist of only letters, digits & underscore.
→ only 31 characters are significant.
→ can not used a keyword for variable name.
→ must not contain white space.

Example :-

Stud 1 ✓	int = 10 ✗	TOTAL ≠ total
Stud-name ✓	Stud name ✗	
flot PI = 3.14 ✓	flot = 3.14 ✗	

(4) String :-

↳ A string is a sequance of character enclosed within double quote.
↳ The characters may be letters digits or special symbol and blank space.

Example :-
"Hello", "123" "well done", "?----!"
'A' ≠ "A"

(5) Special symbol :-

[ ] — square bracket (open, close)
↳ Array dimention to used.

( ) — parenthesis / bracket (open, close)
↳ use for function declaration & function call.

{ } — curly braces
↳ opening & closing of code of main program.

↳ opening & closing of function, If statement & loop block.

, — comma
↳ used for separating a variable and parameter of a function.

— Hash
↳ pre-processor directive.
↳ used for header file.

* — Asterisk
↳ used for pointer & multiplication.

(6) operators :-

↳ An operator is a symbol that tells the computer to perform certain mathematical or loggical manipulations.

↳ operators are used in a program manipulate data variable.

operators
- Unary operators
- Binary operators

Unary operators

a=10

- Increment
  - pre-Increment: ++a
  - post Increment: a++
- Decrement
  - pre-Decrement: --a
  - post Decrement: a--

Binary operators

- Arithmetical
- logical
- Relational
- Assignment
- Conditional
- Bitwise

Simple Program in C

```
#include <stdio.h>
#include <conio.h>
void main ()
{
  clrscr ();
printf ("welcome");
  getch ();
}
```

```
#include <stdio.h>
#include <conio.h>
void main ()
{
  int a;
  clrscr ();
  a=10;
  printf ("%d", ++a);
  printf ("\n%d", --a);
  printf ("\n%d", a++);
  printf ("\n%d", a--);
  printf ("\n%d", a);
  getch ();
}
```

output
11
10
10
11
10

HISTORY Of 'C'

1960	ALWOL	International Group
1967	BCPL	Martin Richards
1970	B	Ken Thompson
1972	Traditional C	Dennis Ritchie
1978	K & RC	Kernighan and Ritchie
1989	ANSI C	ANSI Committee
1990	ANSI/ISOC	ISO committee
1999	C99	Standardigation committee

- C is most popular computer language today because it is a structure and High level machine Independent language.
- C language is use to devoloped a Software, like a stand alone Application
- The root of all modern language is algol interoduce in the early 1960.
- ALWOL was the first computer language to use a block structure.
- it was used in Europe.
- ALWOL gave the concept of structure programming to the computer science community.

- In 1967, martin Richards developed a language called BCPL (Basic Combined programming language) for writing system software.
- In 1970, ken Thompson created a language using many features of BCPL and it is simply called 'B'.
- B was used to create early version of unix operating system at Bell Laboratories.
- Both BCPL and B was typeles system programming language.
- C was ~~evoted~~ evolved from ALGOL, BCPL & B. by Dennis Ritchie at the Bell laboratory in 1972.
- C uses many concepts from these language at added the concept of data types and other powerfull teachers.
- Unix operating system is developed in C language.
- Unix is the one of the most popular Network operating system in use today

and heart of the Internet data super highway.

Backslash constant / Escape sequence

Constant	Meaning
'\a'	Audible alert (bell)
'\b'	back space
'\n'	New line
'\r'	Carriage return
'\t'	horizontal ~~tap~~ tab
'\v'	vertical tab
'\''	single quote
'\"'	double quote
'\?'	question mark
'\\'	back ~~spash~~ space
'\0'	null
'%'	Modular division
'/'	Division

VARIABLE :-

- variable is an Identifier which denotes some allocated space in memory.
- A variable is a data name that may be used to store data value.

→ in variable value can be change during program execution.

eg:- n = 235 , n = 2+3+5 = 10

Name of variable

Average, height, total-Amount, total, first-name

DATA TYPES

- Basic (Primary)
  - char
  - float
  - double
- Derived
  - Array
  - pointers
  - Structure
  - union
- Enumuration
  - enum
- void
  - void

Data types	memory size	Range
char	1 byte	-128 to 127
signed char	" "	" "
unsigned char	" "	0 to 255
Short	2 byte	-32768 to 32767
signed Short	" "	" "
unsigned short	" "	-2^{15} to $+2^{15}-1$
		0 to 65535

Data types	Memory	Range
int	2 byte	-32768 to 32767
signed int	" "	" "
unsigned int	" "	0 to 65535
short int	2 byte	-32768 to 32767
signed short int	" "	" "
unsigned short int	" "	0 to 65535
long int	4 byte	-2,147,483,648 to 2,147,483,647
signed long int	" "	" "
unsigned long int	" "	0 to 4,294,967,295
float	4 byte	3.4e-38 to 3.4e+38
double	8 byte	1.7e-308 to 1.7e+308
long double	10 byte	3.4E-4932 to 1.1E+4932

→ Size :-

1 bit = 0 or 1
0,1 = 1 bit
4 bit = 1 Nible/half byte
8 bit = 1 byte
1024 byte = 1 KB (Kilo byte)
1024 KB = 1 MB (Mega byte)
1024 MB = 1 GB (Giga byte)
1024 GB = 1 TB (Tera byte)

printf ( ) → <stdio.h> → Output → Standard Input output

scanf ( ) → <conio.h> → Input → console Input output

Syntax

⇒ printf ("String format", argument);

⇒ scanf ("String format", argument);

format Speifiers

%d → Integer
%c → character
%f → float
%s → String

```
Q) void main ()
{
    int a=5;
    char c='A';
    float d=3.5;
    Char Name[5]="Sundar"
    printf ("%d\n", a);
    printf ("%c\n", c);
    printf ("%f\n", d);
    printf ("%s\n", name);
    getch ();
}
```

Output
5
A
3.5
Sunda


```
Q) void main ()
{
    printf ("* * * * *\n");
    printf ("* * * *\n");
           ("* * * * *\n");
           ("* * * * *\n");
           ("* * * * *\n");
    getch ();
}
```

```
Q) void main ()
{
    printf ("*\n**\n***\n");
    getch ();
}
```

*
**
***

```
Q) void main ()
{
    int n, s;
    printf ("enter a no:");
    scanf ("%d", &n);
    s = n * n;
    printf ("square = %d", s);
    getch ();
}
```

Square

Q) void main()
{
float n;
printf("~~enter~~ enter a no with fraction:");
scanf("%f", &n);
printf("%d", n);
getch();
}

output
enter no = 3.0
= 3

Q) void main()
{
int n, reverse, sum;
printf("enter a no of three digite:");
scanf("%d", &n);
ld = n % 10;
q = n / 10;
md = q % 10;
fd = q / 10;

reverse = ld * 100 + md * 10 + fd;
sum = ld + md + fd;
printf("reverse = %d", reverse);
printf("sum = %d", sum);
getch();
}

Comments

(1) Single line comment :-

// any comments

(2) Multiline comments :-

/* any comment 1
any comment 2
!
any comment n */

Q) // find the Area & circumference of circle.

main()
{
float r, PI, A, C;
PI = 3.14;
printf("enter radius:");
scanf("%f", &r); // ~~enter~~ input like float t
A = PI * r * r;
C = 2 * PI * r;

printf("in Area = %f", A);

printf("in circumference = %f", C);
getch();
}

ASCII

American standard code for Information Inter change

All country have diffrent type code, but one day meeting and discuse the ASCII code and launch for all country.

```
//ASCII value.

main ()
{
  char c;
printf ("enter a character:");

scanf ("%c", &c);

printf ("ASCII value of %c = %d", c);

  getch ();
}
```


Assignment operators (=)

Statement with simple assignment operator	Statement with shorthand assignment operator
a = a+1	a+ =1
a=a-1	a- =1
a= a * (n+1)	a * = n+1
n= n*i	a / = n+1
	n * = i

Increment & Decrement operator

++(1) | --(1)

post-increment
a=10 ⇒ 10
a++ ⇒ 10
c=a ⇒ 11

post-Decrement
a=5 ⇒ 5
b-- ⇒ 5
c=b ⇒ 4

pre-increment
a=10 ⇒ 11
++a ⇒ 11
c=a ⇒ 11

pre-Decrement
b=5 ⇒ 4
--b ⇒ 4
c=b ⇒ 4

Q) Program :-

```
main ()
{
 int a,b;
  int x,y;
a=10 , b=20;
```

```
x=5, y=8;
a = ++x + ++y;
b = x-- - --- y;
printf("\n %d", a);
printf("\n %d", b);

--a;
--b;
a=b--;

printf("\n %d", a);

--x;
--y;

x= --y;
printf("\n %d", x--);

getch();
}
```

Output

15
-2
-3
5

onditional operator & Ternary operator

ax

```
var = exp? exp2 : exp3
```

nple

```
a=10;
b=20;
greater=a>b ? a:b;
```


Bitwise Operator

Operator	Meaning
&	bitwise AND
|	bitwise OR
^	bitwise exclusive OR (XOR)
<<	left shift
>>	Right shift

~~True~~

Truth Table

A	B	&	|	^
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	1

Loop

1. while → pretesting loop
2. DO---while → post-testing loop
3. for loop

Syntax

```
(1) while (Test condition)
    {
      <body of while>
      <iner/ Decr>
    }
```

example

```
int i=1;
while (i<=5)
  {
    printf("%d", i);
    i++;
  }
```

output

12345

Q) Print the Armstrong.

```
main ()
{
  int n, c, r, sum=0, p;

  printf("enter any no");

  scanf("%d", &n);
    p=n;
  while (n > 0)
  {
    r= n%10;

    c= r*r*r;

    sum = sum + c;

    n= n/10;
  }
  If (P == sum)
   printf(" Armstrong");
  else
  printf(" Not Armstrong");

    getch ();
}
```

Q) print the palindrome.

```
main ( )

int n, r, sum=0, a;
  a=n;
while (n>0)

  r= n % 10;

  sum = sum * 10+r;

  n= n / 10;
}

  If (a==sum)
  printf(" palindrome");
else
  printf (" Not palindrome");
    getch ();
}
```


DO---WHILE

Syntax

(2) Do ---- while

```
DO
{
  <body of do while loop>

  <incr / Decr>

}
  while (expression);
```

Example

```
main ()
{
  int i;
DO
  {
   i=1
  printf("%d", i*i);
    i++;
  }
   while (10>=i);
  getch();
}
```

Output

1 4 9 16 ---- 10th

for loop

Syntax

(3) for-loop

```
for (exp1 ; exp2 ; exp3)
{
 < body of for loop>
}
```

OR,

```
for ( initialization ; condition ; incr/Decr )
{
 <body of loop>
}
```

Example

```
main()
{int i
for( i=1 ; i<=5 ; i++)
{
 printf("%d\n", i);
}
printf("%d", i);
getch();
}
```

Output

1
2
3
4
5
6

Nested loop

Syntax

```
Outer loop
{
 inner loop
 {
  <inner block>
 }
 <outer block>
}
```

Example

```
int i=1, J;
while ( i<=5)
{
 J=1;
while (J<=5)
{
 printf("%d", J);
 J++;
}
 printf("\n");
 i++;
}
```

Output

12 3 45
12345
12345
12345
12345

for loop

...m

...) for_loop

...or (exp1 ; exp2 ; exp3)

< body of for loop>

...,

```
for (initialization ; condition ; incr/Decr)
{
  <body of loop>
}
```

...e

```
main()
{
int i
for( i=1 ; i<=5 ; i++)
{
printf("%d\n", i);
}
printf("%d", i);
getch();
}
```

Output

1
2
3
4
5
6

Nested loop

Syntax

```
Outer loop
{
  inner loop
  {
   <inner block>
  }
  <outer block>
}
```

Example

```
int i=1, J;
while (i<=5)
{
  J=1;
while (J<=5)
{
printf("%d", J);
J++;
}
printf("\n");
i++;
}
```

Output

12 3 45
12345
12345
12345
12345

JUMP STATEMENT

- goto
- break
- continue
- exit (0) function

goto:-

```
void main()
{
  int i, n=2;
  i=1;
top:
printf("%d\n", n*i);
   i++;
If i<=10;
   goto top;
getch();
}
```

output

2
4
6
8
10
12
14
16
18
20

2) break:-

```
void main()
{
  int i;
for (i=1; i<=5; i++)
{
  If (i==3)
    break;
printf("%d", i);
}
  getch();
}
```

output

1 2

3) continue:-

```
main()
{
  int i;
for (i=1; i<=5; i++)
{
  If (i==3)
  continue;
printf("%d", i);
}
  getch();
}
```

output

~~12 45~~
1245

4) Exit (0) :-

```
# include <process.h / stdlib.h>

1. Add
2. substract
3. multiple
4. Divide

switch
{
case1 : Add
<----->

case2 : substract
<----->

case3 : multiple
<------>

case4 : Divide
<----->

case5 : Exit(0)
}
```


ARRAY

- → uses defind data type.
- → it is a sequence of variable.
- → it avoides to declare multiple variable.
- → it is Homogenious nature.

syntax

Data type array name [sige of array];

eg:-

int a [1000];

a	1	2	3	7	5	6	------	1000
	0	1	2	3	4	5		999

2D ARRAY

syntax

int n [sige of row] [sige of colum];

int n [2] [2];

int n [2] [3];

n[0][0] = 5;
n[0][1] = 8;
n[0][2] = 9;
n[1][0] = 2;
n[1][1] = 3;
n[1][2] = 3;

	col0	col1	col2
row0	5	8	9
	00	01	02
row1	2	3	3
	10	11	12

stud [3][2] = {
{123, 80},
{143, 90},
{150, 85},
};

	col0	col1
row0	123	80
row1	143	90
row2	150	85

m2

~~Su~~ Stude[3][2] = {123, 80, 143, 90, 150, 85};

r

e[][2] = {123, 80, 143, 90, 150, 85};

Typedef

→ keyword in c language.

→ user-defined data type.

→ used to define data type in your own language.

* Basic data type:-

int, float, char, String

typedef int num;

num a,b;

typed ~~of~~ in var;

```
int a,b;

var a,b;
(int)
```

function

function

e-difined function
(library f^n)

eg:- printf()
scanf()

use-difined function
eg:- var = sum(10,20)
30

ntax

return-type function name (no of argument);

or (sequence, types);

eg:-
int sum(int, int);
float Area (float, int);
void disp();

arg & NO return	No arg with return	arg with retn	arg with noretn
void fun()	int fun()	int fun(int a)	void fun(char x)
---	{ --- return 5; }	{ --- return 9a; }	{ --- printf("%c",x); }
fun();	int a=fun();	int s=fun(10);	fun('A');

function in c

(1) (No argument & NO return type)

```
#include <stdio.h>
void wel();  //declaration of funct^n
void main()
{
  wel();  // calling funct^n
  getch();
}
funct^n
void wel()  //definition part of funct^n
{
  printf("welcome function");
}
```

(2) (argument & no return type)

```
#include <stdio.h>
void input(int, int);
void main()
{
  int a,b;
  a=10;
```

```
    b=20;

    input (a,b);

      getch ( );
  }
```

funct^n

```
void input (int x, int y)
{
  printf("a=%d ; b=%d", x,y);
  }
```

(3) (NO argument with return type)

```
#include <stdio.h>
float area( );
void main( )
{
  float A;
  A= area( );
printf (" Area of circle= %f",A);
    getch( );
  }
```

funct^n

```
float area( )
{int r=5, PI=3.14;
  return PI*r*r;
  }
```


(4) (argument with return type)

```
#include <stdio.h>
float area ( float ,int);
  void main( )
{
  int r=5;
  float ~~r=5~~ PI =3.14;
  printf("\n Area= %f", area (PI, r));
      getch( );
  }
```

funct^n

```
float area ( float PI , int r)
  {
    float A;
    A= PI * r * r;
    return A;
  }
```

Local & Global variable

<scop of variable>

```
# include <stdio.h>
int x=20;
int a;
void main()
{
int a=25; x=5, y;
printf("\n a=%d", a);
printf("\n x=%d", x);
printf("\n y=%d", y);
{
printf("\n x=%d", x);
printf("\n a=%d", a);
}
getch();
}
```

output

a= 25
x=5
y=Gov
x=20
a=0

STORAGE CLASSES

1) auto → local → Gov (by default)

2) extern → Global → zero (by default)

3) Static → fixed (one time initialization) → 0 (by

4) resister → fast execution → Gov (by default)
(local)

(1) auto :-

```
main ()
{
auto int a=10; // local
printf("a=%d", a);
}
```

If here is nothing value of a then output will be Gov

output

a=10

(2) Extern :-

```
int a=10;
main()
{
int b=20;
extern int a;
printf("a=%d ; b=%d", a,b);
}
```

If here extern is not difine then a=0 (by default)

output

a=10, b=20

(3) Static :-

```
main()
{
Static int a=10;
printf("a=%d", a);
a=20
printf(" a=%d", ++a);
}
```

output
a=10
a=11

(4) resister :-

```
main()
{
resister a;
printf(" a=%d", a);
a=20;
printf(" \n a=%d", a);
}
```

output
a=garbage
a=20

POINTER

→ pointer is a variable, that store the ~~pointer is a variable~~ address of another variable.

→ pointer holds the address of a variable for example an Integer variable holds an Integer value, However an Integer pointer holds the address of a Integer variable.

example

int var → [5] → value / constant

& var → [2000] → address

scanf("%d", &var);

pointer syntext

data-type — *variable-name,

OR

data type — variable-name

example

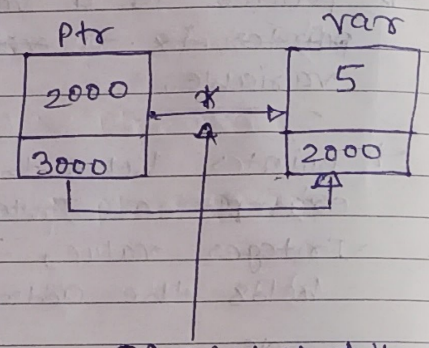
int *p; OR int* p_1, p_2

pointer

normal variable

Assigning adress to pointer

```
int *ptr, var;
var=5;
ptr = &var;
```



If point that then store the value(5)

Getting pointed value by pointer

```
int *ptr, var;
var =5;
ptr =&var;
printf("%d", *ptr); → 5
```

Changing value pointed by the pointer

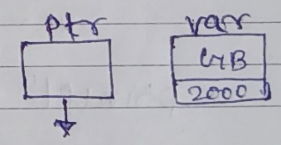
```
int *ptr, var;
var=5;
ptr = &var;
var=10;
printf("%d", var); → 10
printf("%d", *ptr); → 10
```


Example

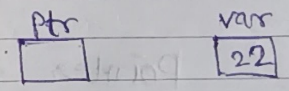
```
int *ptr, var;
var= 5;
ptr = &var;
*ptr = 10;
printf("%d", *ptr); → 10
printf("%d", var); → 10
```

Explaination

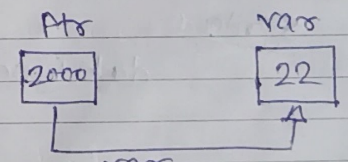
1) `int *ptr, var;` →



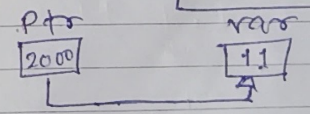
2) `var = 22;` →



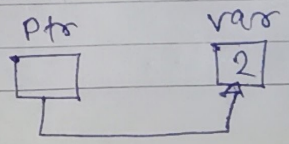
3) `ptr = &var;` →



4) `var = 11;` →



5) `*ptr = 2;` →



Comman Mistake

```
int var, *ptr;

ptr = var   // error -> ptr = &var

*ptr = &var  // error -> ptr = &var
```

Confusion

```
int a=5;

int *p = &a;  <-->  int *p = &a;
                    int *p;
                    p = &a;   equal

printf("%d", *p); -> 5
```

Pointer to pointer (Double pointer)

Syntax

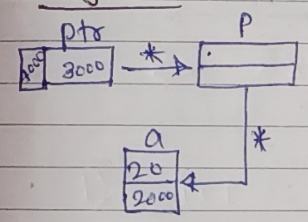
datatype ** variable-name

underscore
or
underline

Example-I

```
int **ptr;
int *p;
int a=20;

p = &a;
ptr = &p;

printf("%d", a); -> 20
printf("%d", *p); -> 20
printf("%d", **ptr); -> 20
```

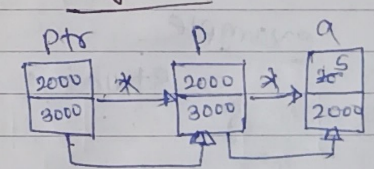
Diagram



Example-II

```
int **ptr;
int *p;
int a=20;
p = &a;
ptr = &p;
printf("%d", **ptr); -> 20
*ptr = 5;
printf("%d", *ptr); -> 5
printf("%d", *p); -> 5
printf("%d", p); -> 2000
printf("%d", a); -> 5
```

Diagram



ASCII

- American standard code for information Interchange.
- All country have diffrent type code, but one day meeting and discuse the ASCII code and lonch for all country

Note:- it is already available in copy Book

Structure in C

syntax

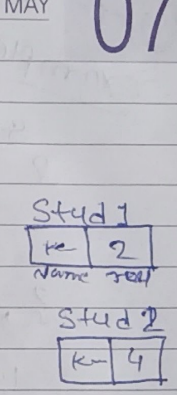
```
Struct tag-name
{
  data member declaration
};
```

example

```
Struct employee
{
  int emp-ID;
  char emp_name[50];
  double emp-salary;
} emp;
```

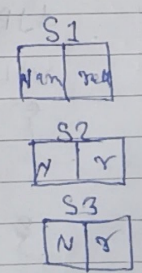

example 1

```
Struct Student
{
    int roll;
    char name[50];
};

main()
{
    struct Student Stud1;
    struct student Stud2;
}
```



example 2

```
Struct student
{
    int roll;
    char name[50];
};
typedef struct student stud;

void main()
{
    stud S1, S2, S3;
}
```



example 3

```
struct student
{
  int roll;
  char name[50];
};

typedef struct student stud;

void main()
{
  stud s1={7,"s"};
  stud s2;
}
```

example 4

```
struct student
{
  int roll;
  char name[50];
};
typedef struct student kush;
void main()
{
  ~~stude~~ kush s1={.roll=8, .nam="k"};
}
```

.member
→ pointer

example 5

```
typedef struct student
{
  int roll;
  char name[50];
} stud;
void main()
{
  stud s1, s2;
}
```

Array with Structure

example

```
structure k
{
  int roll;
  char name[20];
};
void main()
{
  structure k s1[5];
  int i;
  printf("enter roll ;name of 5 student");
  for (i=0; i<5; i++)
  {
  printf("Enter roll:");
```

```
scanf("%d", &s1[i].roll);
printf("\n Enter name:");
scanf("%s", &s1[i].name);
}
}
```

Pointer with structure

example

```
structure K
{
  int x,y;
};
void main()
{
  struct K s1, *sp;
  s1.x = 10;
  s1.y = 20;
  sp = &s1;
printf("x=%d", *sp->x);
printf("y=%d", *sp->y);
}
```


function with structure

example

```
void fn (struct st s);
struct st
{
  int x,y;
};

void main()
{
  struct st s1;
  s1.x = 10;
  s1.y = 20;
  fn (s1);
}

void fn (struct st s)
{
  printf("x=%d", s.x);
  printf("y=%d", s.y);
}
```

Union

```
union K
{
 int roll;
 char name[5];
};
void main()
{
  union K S1;
printf("%d", sizeof (S1));
printf("%c", S1.name);
}
```

Note:- union store the longest size of memory and it store many types of int and float, char in a single longest size memory.

Enumaration data type

Syntax

Enum enum name {const1, const2 --- const n};

Example

```
Enum weak {"sun", "mon", "Tue", "wed", "Thu", "fri", "sat"};
for (i=sun; i<=sat ; i++)
{ printf("%d", i);
```

output

0, 1, 2, 3, 4, 5, 6

DMA

(Dynamic memory Allocation)

(1) Malloc () → single block

(2) calloc () → ~~Multi~~ multiple block (array)

(3) realloc () → increase / Decrease the sizeof allocated memory

(4) free () → deallocate the allocated memory.

(1) Malloc ():- it is build in function.
→ Header file (stdlib.h).
→ it is short name for memory allocation and used to dynamic allocation a single large block of contiguous memory according to the size specify.

Syntax

(void *) malloc (size_t size)

void pointer, unsigned, byte

example

int *ptr = (int *) malloc (4);

or/

int *ptr = (int *) malloc (10* sizeof (int));

(2) Calloc ():- It is used to dynamically allocat multiple block of memory.

→ It is diffrent from malloc in two way,

(i) calloc needs two argument

syntax

void * calloc(size_t n, size_t size)

No of block

size of each block

example

int *ptr = (int *) calloc (10, sizeof (int));

→ An equivalent malloc

example

int *ptr = (int *) malloc (10 * sizeof (in));

Note :-

→ memory allocated by calloc is Intiallized to 0 (zero).

→ memory allocated by malloc is Intiallized with some garbage value.

(3) realloc ():- it is used to change the size of memory block without losing the old data

syntax

(void *) realloc (void * ptr, size_t new size

example

int *ptr = (int *) malloc (sizeof (int

ptr = (int *) realloc (ptr, 2* sizeof (

(4) free () :-

→ free function is used to realleas the dynamically allocated mem in Heap.

Syntax

void free (ptr);

example

```
main()
{
  int *ptr = (int *) malloc (5 *
  ⋮
  free (ptr);
  getch();
}
```

#include <string.h>

String

→ To way of declaration of string.

(1) By char Array → char str1[]={'w''E''L''c''o''M''E'};

(2) By string literals → char str2[] = "welcome";

example

```
char str1[] = {'w''E''L'};
char str2[] = "WEL";
printf("%s", str1); → WEL
printf("%s", str2); → WEL
printf("%c", str1[3], str2[1]); → LW
```

String functions

(1) strcpy();
(2) strlen();
(3) strcat();
(4) strcmp();
(5) strupr();
(6) strlwr();
(7) strrev();
etc----

(1) strcpy(); :— copy the string.

example

```
char str1="abcd", str2="abc"
strcpy(str1, str2);
puts(str1); → abc
puts(str2); → abc
```

(2) strlen(); :— it returns the length of string.

example

```
int len;
char str1[20];
strcpy(str1, "welcome");
len = strlen(str1);
printf("%d", len); → 7
```

(3) strcat(); :— it is concantinent fun which is used to join the two string.

eg:-

```
char str1[] = "This is";
char str2[] = " program";
strcat(str1, str2);
puts(str1); → This is program
puts(str2); → program
```

(4) strcmp ( ); :- compare two string.

Return type

zero(0) :- If str1 and str2 is equals.

Negative → If str1's ASCII value is less than str2's ASCII value.

positive → If str1's ASCII value is greater than str2's ASCII value.

example

```
char str1[] = "abcd", str2[] = "abCd",
      str3[] = "abcd";
```

(67-99)
(C - c) = ↓

```
printf("%d", strcmp(str1, str2)); → 32
printf("%d", strcmp(str1, str3)); → 0
```

(5) strupr(); :→ lowecase to upercase.

eg:-

```
char str1[] = "abcd";
strupr(str1);
puts(str1); → ABCD
```


(6) strlwr (); :- upercase to lowercase

eg:-

```
char str[] = "KUSH";
strlwr (str);
puts (str); → kush
```

(7) strrev(); :- reverse the string.

example:

```
char str[] = "kush"
strrev (str);
puts (str); → hsuk
```

Q) Reverse without string function :-

```
main()
{
    int len=0; i, s=0, j=0;
    char str[20], rev[20];
    puts("enter any string");
    gets (str);
```

```
for (i=0; str[i]!='\0'; i++)
    len++;

for (i= len-1; i>=0; i--)
{
    If (str[i] == str[j++])
        s =1;
    else
        break;
}
    If (s==1 && i==(len-1))
    puts ("palindrom.");
else
    puts ("Not palindrom.");
    getch();
}
```


file Input / output in C

fputc — writing in file

fgetc — read from file

"r" — read mode

"w" — write mode

"a" — append mode

```
FILE *fp;
fp = fopen ("file.txt","w"); // wri
fp = fopen ("file.txt","r"); // rea
fp = fopen ("file.txt","a"); // aper
fclose (fp);
```

file opening mode

"r" → it searchess file If file is opened successfully fopen lods into memory and sets of a pointer.

— If file cannot be opened fopen return the NULL value

"w" → it searchess the file is exisists If contents are over written. If ~~contents~~ file doesn't exit, A New file is created. it return NULL If enable to return NULL, 0

"a" → it searchess file. it open the file successfully. fopen ( ) writes it into memory and sets a pointer that's point to the last character in it. If the files don't exit new file is created. it also returns 'NULL'. it enable to open file. This is use to ~~open~~ adding new content at new contents at end of file (EOF).

File operation

1. creation of new file.
2. opening an exiting ~~file~~ file.
3. Reading from a file.
4. writing to a file.
5. Moving to specific location in f
6. closing a file.

* Creation of New file and write

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

void main ( )
{
    int i, s;
    char n [24], c;

    FILE *fp;

    clrscr( );
    fp = fopen ("stud.txt", "a");
```

```
If(fp = = NULL)
    printf(" \n Can't open file");
else
    printf(" \n write something");
gets(n);
s= strlen(n);
for (i=0; i<=s; i++)
    fputc(n[i], fp);
fclose (fp);
getch();
}
```

Method II

27 SUNDAY ~~Reading~~ **writing in file**

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



```
void main()
{
    char c;
    FILE *fp;
fp= fopen(" file.txt", "w");
while (1)
{
    c = getc();
If (c == EOF)
    Break;
put (c);
}
fclose (fp);
getch();
}
```

Reading from file

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

void main()
{
    FILE * fp;
    Char ch;
fp = fopen ("file.txt", "r");
while (1)
  {
    ch= fgetc (fp);
    If (ch==EOF)
      break;
    printf ("%c", ch);
  }
fclose (fp);
getch();
}
```