(6) User defined data types in C programming.

The user defined data types are basically designed by us to fulfil our needs. There are four types of identifiers allows in c language :—

(I) Structures
(II) Union
(III) Typedef
(IV) Enum

(I) Structures — We use the structure for organising a group of various data items into a single entity — such as the sets of information regarding a person or an account etc. Every data item present in a structure is known as a member. These members are also called fields. keyword — struct is used to create a structure.

Syntax is :-

```
struct name_of_structure {
    data_type variablename;
    data_type variablename;
    - - - - - -
    - - - - - -
};
```

(II) Union — We can define union by using union keyword but at any given time, just a single member could be able to contain the given value. Union stores different data type variables in the same memory location.

Structure and unions are very similar to each other. The only difference is that we can access just a single member of the union at any given time. It is because the creation of memory is only for one member that has the biggest size.

Syntax is :-

```
union name_of_union {
    datatype variablename;
    datatype variablename;
    - - - - -
    - - - - -
};
```

(III) typedef — We can use the keyword typedef for creating an alias (a new name) for a data type that already exists. The typedef would not create any new form of data type.

Syntax :- typedef existing_data_type new_type;

(7)

(iv) enum — enum is a keyword which is used to create an enumerated data type. It is specially a new datatype that consists of a set of various named values — known as members or elements. We mainly use it for assigning different names to the integral constants. This way, a program becomes much more readable.

Syntax:- enum identifier (value_a, value_b, ...);

The enumerated data types allow a user to create symbolic names of their own for a list of all the constants that are related to each other.

eg:- enum Boolean { true, false };

enum week { MON, TUE, WED, THURS, FRI, SAT, SUN };

Example:-
```
#include <stdio.h>
int main(){
    enum week {MON=1, TUE, WED, THURS, FRI, SAT,
               SUN};
    enum week off=SUN;
printf("week off = %d", off);
    return 0;
}
```

Output will be :- Week off = 7

sizeof() operator :- We can use sizeof() operator in C language so that we can determine the sizes of the data types or the expressions that are specified in the storage units of char-size. Important point to be noted that — the output obtained due to sizeof() operator varies a lot on the basis of different types of machines. For example, a 32-bit OS (operating system) and a 64-bit operating system would generate very different results for the same data type in a program.

```
double x=78.0;
float y=6.78;
printf("\n Size of x=%d", sizeof(x));
printf("\n Size of x+y=%d", sizeof(x+y));
```

⑥ Bitwise operators in C programming

There are some bitwise operators used in C programming. There are following 6 bitwise operators used in C programming—

- Bitwise OR operator — |
- Bitwise AND operator — &
- Unary Operator (Binary one's complement) — ~
- Bitwise XOR operator — ^
- Binary Right Shift operator — >>
- Binary Left Shift operator — <<

Suppose we have two integer variables as : —

```
int P = 60;
int Q = 13;
```

P | Q ≈ OR 111100 / 001101 = 111111

(P) 60 → 111100

2	60	0
2	30	0
2	15	1
2	7	1
2	3	1
	1	

(Q) 13 → 1101

2	13	1
2	6	0
2	3	1
	1	

$(111101)_2 = 1\times2^5 + 1\times2^4 + 1\times2^3 + 1\times2^2 + 0 + 1\times2^0$

$= 32 + 16 + 8 + 4 + 1 = 61$

P & Q ≈ AND 111100 / 001101 = 001100 = 1100

$= 1\times2^3 + 1\times2^2 + 0 + 0$

$= 8 + 4 = 12$

$(\sim P) = \sim(60) = \sim(111100) = 000011$

$= 3$

P ^ Q ≈ ^ 111100 / 001101 = 110001

>> (Right shift) operator moves the value of the left operand to the right by the number of bits that the right operand specifies.

P >> 2 = 15 which is 001111

<< (Left shift) operator moves the value of the right operand to the left by the number of bits that the right operand specifies. P << 2 = 240 which is 11110000

Now, let us understand by examples:—

```
#include <stdio.h>
    int main(){
        unsigned int P=60;
        unsigned int Q=13;
    int r=0;
    r= P|Q;
printf("\n %d | %d = %d", P, Q, r);
    r= P & Q;
printf("\n %d & %d = %d", P, Q, r);
    r= ~P;
printf("\n ~ %d = %d", P, r);
    r= P^Q;
printf("\n %d ^ %d = %d", P, Q, r);
    r= P >>2;
printf("\n %d >> 2 = %d", P, r);
    r= P <<2;
printf("\n %d << 2 = %d", P, r);
  return 0;
  }
```

Compile and Run the program by giving different values to variables P and Q and see the results.

Note:— ① Right shift operator and Left shift operator must not be used with -Ve numbers.

② We can not use bitwise operators in the place of any logical operator (&&, ! and ||).

The result that we get from a logical operator (&&, ! and ||) is either 1 or 0. But the bitwise operators, on the other hand, return a value that is an integer.

Example:—

```
#include <stdio.h>
    int main(){  int a=2, b=5;
    (a & b) ? printf("False\n"): printf("True\n");
    (a && b) ? printf("False\n"): printf("True\n");
        return 0; }
```

Compile & Run the above program will print the result as:—

```
False
True
```

Hence, we cannot use && in place of & operator, because (binary)
result will be printed wrong. (logical)

The right-shift and left-shift operators are equivalent to respectively division and multiplication by 2. But it will work only when the available numbers are positive. Let us see following example:—

```
#include <stdio.h>
int main() {
int a = 19;
printf("a >> 1 = %d\n", a >> 1);
printf("a << 1 = %d\n", a << 1);
return 0;
}
```

The output of the above program would be:—

```
a >> 1 = 9
a << 1 = 38
```

We can also use the & operator to check quickly if a number is even or odd.

Example:—

```
#include <stdio.h>
int main(){
    int x;
printf("\nEnter a number:"); scanf("%d", &x);
(x & 1) ? printf("%d is even", x):
        printf("%d is Odd", x);

return 0;
}
```

Run the program and Enter a number to check for even or odd.