UNIT -2

INPUT AND OUTPUT STATEMENTS, OPERATORS AND EXPRESSIONS IN C

Introduction

Reading processing and writing of data are the three essential functions of a computer program.
Most programs take some data as input and display the processed data as output.
One method is to assign values to variables through the assignment statements such as x = 10; y = 11;.
Another method is used is to use the input function **scanf** ,which reads the data from keyboard and **printf** is used to send results out to a terminal.
Each program that uses a standard input/output function must contain the statements **#incliude<stdio.h>.**
The file name **stdio.h (**standard input –output header file).The instructor #include<stdio.h> tells the compiler to search for a file named stdio.h and place its contents at this point in the program.

Importance of Pre Processor Directive #include

The pre processor is a program that processes the source the program before it is passed on to the compiler. It operates under the control of pre processor directives which began with the symbol '#'. The program is typed in the editor is the source code to the pre-processor.
C supports two pre-processor directives. They are : #include and #define.
There are 3 types of pre processor directives:

1. Macro
2. File inclusion
3. Compiler control

Macro : Macro replaces every occurrence of identifier by a predefined string

Syntax : #define Identifier String
Eg: #define PI 3.14
Instead of writing 3.14 in program we just write PI.

```
#include<stdio.h>
#define LIMITS 5
int main ( )
{
for(int  i = 0; i<LIMITS; i++)
{
printf("%d\n",i);
return 0;
}
}
```

Output : 1
2
3
4

File inclusion

For the inclusion ,we can use the #include.

- We can replace the content that is comprised in the filename at the point where a certain directive is written.
- We can use a file inclusive directive and included the header files in the programs.
- We can integrate function declaration, macrons, and declarations of the external variables in the top header file rather than repeating each logic in the program.
- The stdio.h header file includes the function declarations and can provide the details about the input and output.

The preprocessor directive tells the compiler to include a file in the source code program.#include is used to include the header files.
Syntax : #include <file name > or #include " file name"

Standard header file

#include <file name>: In this example, we can define a certain directory, and we search a file within it.
< > brackets tells the compiler to look for the file in the standard directory

User defined header file

When program becomes very large ,It divides the program into smaller files and include then whenever needed.
Syntax : #define " file name".
Double quotes tells the compiler to search for the header file in the source file directory.

Compiler control or conditional

These are helps to compiler a specific portion of the program or to skip the compilation of some specific part of the program based on the conditions.
It utilizes directives like #if, #elif, #else, and #endif.

Input /Output statements in 'c' language:

- Input and output statements are used to read and write the data in 'C' programming .
- These are embedded in "stdio.h" (ie), standard input output header file.
- They are categorized into two types. They are:

  - ✓ Unformatted Input / Output functions
  - ✓ Formatted Input / Output functions
  - ✓

Unformatted Input / Output functions : These functions do not use any kind of strings to read and write data.

Unformatted Input functions (or) Reading a character using getch(), getche(),getchar()

1. **getchar( )** :
   - It is an input function. It is used for reading a single character from the keyboard.
   - It is a buffered function. It means these functions get the input from the keyboard and store it in the memory buffer temporarily until you press enter key.
   - After pressing the enter key the input moves to relevant variable.

   Syntax : int getchar (void);

   Program

```
#include<stdio.h>
int main ( )
{
printf("%c",getchar());
return 0;
}
```

   Output : g
   g

2. **getch():**
   - it is also an input function used to read a single character from the keyboard like getchar function. But getchar is buffered function and getch is a non buffered function.
   - The character read by this function will be directly assigned to a variable.
   - Another use of this function is to maintain the output on the screen till you have not press the enter key.
   - getch() is non standard function present in conio.h.

   Syntax : int getch(void);
   Program

```
#include<stdio.h>
#include<conio.h>
int main ( )
{
printf("%c",getch());
return 0;
}
```
Output : g

3. **getche():**
   - it is similar to getch function. It means when you type the character data from the keyboard , it will be visible on the screen.
   - getche function also non standard function present in conio.h.

syntax : int getche(void);

program

```
#include<stdio.h>
#include<conio.h>
int main ()
{
printf("%c",getche());
return 0;
}
```
Output : aa

4. **gets():**
   - It is an input function used to read a string from the keyboard.
   - It is also a buffered function. It will read a string, when you type the string from the keyboard and by pressing the enter key from the keyboard.

Syntax : int gets(variable);

Unformatted output functions (or) writing a character using putch(),putchar()

1. **putch():**
   - It is an output function. It is used to displays any alphanumeric characters on the screen.
   - It is non standard output function available on conio.h

```
#include<stdio.h>
void putch(char c)
{
putchar(c);
}
int main ( )
{
char ch = 'b';
putch (ch);
return 0;
}
```

2. **putchar():**
   - It is an output function. It is used to display a single character on the screen.
   - It is a standard function.

```
#include<stdio.h>
int main ()
{
char ch = 'a';
putchar(ch);
return 0;
}
```

3. **puts():**It is an output function used to display a string which is read by gets function on the screen.

Formatted Input / Output functions : These functions use format strings to read and write data.

Formatted Input function scanf()

Scanf():It is an input function and it is used to read the mixed type of data from the keyboard. The general syntax is as follows:

scanf("controlstring",&v_1,v_2,……&v_n);

where v_1,v_2……v_n are all variables. Here '&' specifies the address of location where the data is stores.

Ex: scanf("%d%d",&a,&b);

Control string have some format codes (or) format specifies for different data types.

Commonly used scanf format codes:

%c - Reads a single character.
%d - Reads a decimal integer.
%e or %f - Reads a floating point value.
%h - Reads a short integer.
%i -Reads a hexadecimal, decimal, octal integers.
%o - Reads an octal integer.
%s - Reads a string.
%u -Reads a unsigned decimal integer.
%x -Reads a hexa decimal integer.
%[..]- Reads a string of word.

Formatted output function printf()

printf() : it is an output function .It is used to display a text message and to display mixed type of data on the screen. The general syntax is:

printf("control string",v1,v2,…..v_n);

printf("messageline");

where v1,v2, v_n are called variables. The variables should match in number order and type within the format specifications.

Ex: printf("%d%d",a,b);

Printf("hello world");

Example program on Formatted input / output functions:

```
#include<stdio.h>
int main ()
{
 int num;
printf("Enter a number:");  // printing a message on the output screen.
scanf("%d",&num);  //Taking a number value from the keyboard.
printf("Entered value is :%d",num);  //Displaying the entered value.
return 0;
}
```

Character functions in c

C language provides a large collection of character functions. Character functions are deal with single character at a time.

Definition :

Character functions are the functions which are used to perform the specified operations on the character data type values.

For example, a is represented as 'a' which is actually the integer value equal to 97 in ASCII. Similarly , '\n' represents the integer value of newline equal to 10 in ASCII.

C provides standard character handling library <ctype.h>. We should include character handling library for using different character handling function.
Some character functions are used to check for the character specifications and returns true(1) or false (0).

1. Isalpha( ) : checks whether the character variable contains an alphabet or not.
2. Isdigit ( ) : checks whether the character variable contains digit or not.
3. Isalnum () : checks whether the character variable contains an alphabet or digit.
4. Ispunct ( ): checks whether the character variable contains punctuation or not .
5. Isspace () : checks whether the character variable contains space or not.
6. Isupper ( ) : checks whether the character variable contains uppercase or not.
7. Islower ( ) : checks whether the character variable contains lowercase or not.
8. toupper(): checks whether the character variable contains alphabet or not and convert to upper.
9. tolower(): checks whether the character variable contains alphabet or not and convert to lower.
10. Isgraph(): check whether it is a graphical character or not.
11. Isprint(): check whether printable character or not.
12. Iscntrl(): check whether it is control character or not.
13. Isxdigit():check whether it is hexadecimal or not.

PROGRAM

```
#inculd<stdio.h>
int main()
{
char ch = 'a';
printf("%d",isalpha(ch));
return 0;
}
```

Operators

An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate date and variables.

Types of operators:

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment and Decrement operators
6. Conditional operators
7. Bitwise operator
8. Special operator

Arithmetic Operator

This operator used for numeric calculation. And these operators are addition, subtraction, multiplication, division. Binary arithmetic operator on other hand required two operand and its operators are +(addition), -(subtraction), *(multiplication), /(division), %(modulus). But modulus cannot applied with floating point operand as well as there are no exponent operator in c.

Symbol	Name
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

Program

```
#include<stdio.h>
int main ( )
{
int a = 20, b = 10;
printf("%d\n",a+b);          output : 30
printf("%d\n",a-b);                   10
printf("%d\n",a*b);                   200
printf("%d\n",a/b);                   2
printf("%d\n",a%b);                   0
return 0;
}
```

Relational Operator

It is use to compared value of two expressions depending on their relation. Expression that contain relational operator is called relational expression.

The value or result of a relational expression is either **true** or **false**

Ex: 45>58 False(0)

34==24+10 True(1)

Operator	Meaning
<	less than
>	greater than
<=	less than or equal
>=	greater than or equal
==	equal to
!=	not equal to

Program

```
#include<stdio.h>
#include<math.h>
int main ()
{
int a = 10,b = 20;
printf("%d\n",a<b);      Output : 1
printf("%d\n",a>b);               0
printf("%d\n",a<=b);              1
printf("%d\n",a>=b);              0
printf("%d\n",a==b);              0
printf("%d\n",a!=b);              1
return 0;
}
```

Logiacal Operator

Operator used with one or more operand and return either value zero (for false) or one (for true). The operand may be constant, variables or expressions. And the expression that combines two or more expressions is termed as logical expression.

C has three logical operators :

Operator	Meaning	Example
&&	AND	(a>b)&&(a>c)
||	OR	(a>b)||(a>c)
!	NOT	!(a>b)

Where logical NOT is a unary operator and other two are binary operator. Logical AND gives result true if both the conditions are true, otherwise result is false. And logial OR gives result false if both the condition false, otherwise result is true.

Exp 1	Exp 2	AND
T	T	T
T	F	F
F	T	F
F	F	F

Exp 1	Exp 2	OR
T	T	T
T	F	T
F	T	T
F	F	F

NOT	
Exp 1	Exp 2
F	T
T	F

Program

```
#include<stdio.h>
int main ( )
{
int a = 10, b = 20, c = 30;
printf("%d\n",(a>b)&&(a<c));                    Output : 0
printf("%d\n",(a>b)||(a<c));                             1
printf("%d\n",!(a>b));                                   0
return 0;
}
```

Assignment Operator

Assignment operator are used to assign the result of an expression to a variable. Operand on the left hand side should be variable and the operand on the right hand side should be variable or constant or any expression. When variable on the left hand side is occur on the right hand side then we can avoid by writing the compound statement.

For example : int x= y;
Int Sum=x+y+z;

Program

```
#include<stdio.h>
int main ( )
{
int  a = 10;
printf("%d\n",a);                          Output : 10
printf("%d\n",a+=10;);                              20
return 0;
}
```

There are two types :

1. **Simple : (=)**
2. **Compound : (+=, -=, *=, /=, %= ,<<=, >>=,&=, !=,^=)**

Example for simple : a = 5;
Example for compound : a+=10

a = a + 10
= 10 +10
a = 20

Nested Assignment

If more than one assignment operator is present in statement , then that statement is called nested assignment statement.

Syntax : var1 = var2= var3= var4=……= value or expression .

Ex: i=j=k=m=10;

Increment and Decrement Operators

The Unary operator ++, --, is used as increment and decrement which acts upon single operand. Increment operator increases the value of variable by one .Similarly decrement operator decrease the value of the variable by one. And these operator can only used with the variable, but cann't use with expression and constant .

Increment operators

It again categories into prefix (or) pre-increment and post fix (or) post-increment.

Pre-increment :

Increment operator is placed before the operand and value is first incremented and then operation or expression is performed on it.

Ex: z =++a
a= a+1
z = a

Post- Increment :

Increment operator is placed after the operand and value is incremented after the operation is performed on it.

Ex : z=a++
z=a
a= a+1

Program

```
#include<stdio.h>
int main ( )
{
int a = 10, z;
z=++a;
printf("Z=%d\n",z);
print("a=%d\n",a);
return 0;
}
```

Output : Z = 11
a = 11

```
#include<stdio.h>
 int main ()
{
 int a = 10,z;
 z=a++
 printf("Z= %d\n",z);
 printf("a=%d\n",a);
return 0;
}
```

Output : Z = 10
a= 11

Decrement Operators

It is used to decrement the value by 1. It is categories into pre decrement and post decrement.

Pre decrement

Decrement operator is placed before the operand and the value is first decremented and then operation or expression is performed on it.

Ex: z=--a
a=a-1
z=a

Post Decrement

Decrement operator is placed after the operand and the value is decremented after the operation.

Ex: z=a—
z=a
a=a-1

Program

```
#include<stdio.h>
int main ()
{
```

```
#include<stdio.h>
 int main ()
{
```

```
int a= 10, Z;
Z= --a;
printf("Z=%d\n",Z);
printf("a=%d\n",a);
}
Output : Z=9
         a =9
```

```
int a=10, Z;
Z=a--;
printf("Z=%d\n",Z);
printf("a=%d\n",a);
}
Output : Z= 10
         a= 9
```

Conditional Operator

It sometimes called as ternary operator. Since it required three expressions as operand and it is represented as (? , :).

SYNTAX

exp1 ? exp2 :exp3

Here exp1 is first evaluated. It is true then value return will be exp2 . If false then exp3.

EXAMPLE

```
#include<stdio.h>
 int  main()
{
int  z;
 z = (5>3) ? 1 : 0 ;
printf("%d",z);
 }
```

Output: Value is:1

Bitwise Operator

Bitwise operator permit programmer to access and manipulate of data at bit level. Various bitwise operator enlisted are

one's complement	(~)
bitwise AND	(&)
bitwise OR	(|)
bitwise XOR	(^)
left shift	(<<)
right shift	(>>)

Program

```
#include<stdio.h>
int main ()
{
int a= 12,b = 10;
printf("%d\n",a&b);
printf("%d\n",a|b);
printf("%d\n",a^b);
return 0;
}
```

Output : 8
14
6

Bitwise AND : The **& (bitwise AND)** in C takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.

Bitwise OR : The **| (bitwise OR)** in C takes two numbers as operands and does OR on every bit of two numbers. The result of OR is 1 if any of the two bits is 1.

Bitwise XOR : The **^ (bitwise XOR)** in C takes two numbers as operands and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.

a = 12 , b = 10

binary notations for a and b by using 8 4 2 1 bit level and based on truth tables.

a= 12 1100

b=10 1010
a&b = 8 1000
a|b = 14 1110
a^b=6 0110

Bitwise left Shift : The **<< (left shift)** in C takes two numbers, the left shifts the bits of the first operand, and the second operand decides the number of places to shift. If the values of a variable is left shifted one time, then its value get doubled. When Data is Shifted Right ,leading zero's are filled with zero.
Ex: a = 10 then a<<1 = 20.
Note : left shifting is equivalent to multiplication by $2^{\text{right operand}}$.
Ex: 10 * 2^1 = 20
$10*2^4$= 160.
Bitwise Right Shift : The **>> (right shift)** in C takes two numbers, right shifts the bits of the first operand, and the second operand decides the number of places to shift. If the values of variable is right shifted one time, then its value becomes half of the value.
Ex: a = 10 then a>>1 = 5.
Note : right shifting is equivalent to division by $2^{\text{right operand}}$.
Ex : $10/2^1 = 5$
a = 32 then a>>4
$32/2^4$ = 32/16 = 2.
Bitwise One's Complement : The **~ (bitwise NOT)** in C converts all one's to zero's and all zero's to one's.
Ex: a=5 then ~a =- 6.
Note : one's complement of any number is **N is –(N+1).**
Two's complement of any number is equal to complement of that number + 1. **~N = -(~(~N)+1).**
Ex: a =5 => 0 1 0 1
~a= 10=>1 0 1 0
~(~a)= 5=> 0 1 0 1 + 1 then by the formula .
~a=-(~(~a)+1)
~a= -(~(10)+1)
~a=-(5+1)
~a= -6.

Program

```
#include<stdio.h>
int main ()
{
int a= 20, b = 10, c = 10;
printf("%d\n",a<<1);
printf("%d\n",b>>1);
printf("%d\n",~c);
return 0;
}
```

Output : 40
5
-11

Special Operator

C supports some special operators such as

1. Comma operator
2. Size operator
3. Pointer operator
4. Member selection operator
5. Ampersand(&)

Comma Operator

When number of statements occur in a 'C' program having a relationship between expressions, So , by using comma operator , we can write all the statements in a single statement using comma operator.
Comma operator is used as Separator:
Ex: int a = 3, b = 4, c = 9;
Comma is used as an operator
int a =(3,4,8);
printf("%d",a);
here the output is 8
comma operator returns the right most operand in the expression and evaluates remaining value and reject them . parenthesis has highest precedence so comma will evaluated.

Comma Operator having least precedence among all the operators available in c.
int a ;
a= 3,4,8
printf("%d",a);
here out put is 3
Note: here another operator (=) it is like (a=3),4, 8 so it returns 3.

Size of operator

It is also called compile time operator. It displays number of bytes(or) size assigned by a variable.
Ex: int n = sizeof(variable);

Pointer operator

*(asterisk) is the pointer operator that is used to represent a pointer variable.
Ex: int*p;

Member selection operator(. and ->)

These operator are used in structures and unions. Dot(.) and point to(->) operators are called member selection operators.

Program

```
#include<stdio.h>
int main ()
{
int a =10;
float b =20;
printf("a=%d b=%f\n",a,b);
printf("Address of a=%u\n"&a);
printf("Address of b=%u\n",&b);
printf("size of a=%d\n",sizeof(a));
printf("size of b=%d\n",size of(b));
}
```

Output:

a= 10, b = 20.0
address of a = 2927348140
address of b =292734136
size of a = 4
size of b= 4

Operator Precedence and Associativity

Precedence

Precedence simply means that which operation is going to perfom first and which is going to perform last in that expression.

Associativity

It is used only when two or more operators are of same precedence. Either left to right or right to left.
For example, multiplication and division arithmetic operators have same precedence, let's sy we have an expression 5*2/10, this expression would bu evaluated as (5*2)/10 because the associativity is left to right for these operators.Similarly 20/2*5 would be calculated as (20*2)/5.

Precedence	Description	Operator	Associativity
1	Function call Array subscript Dot operator(structure & union member access) Arrow operator(structure & union member access through pointer) Suffix/ postfix increment and decrement	( ) [ ] . -> ++,--	Left to right
2	Prefix increment and decrement Unary plus and minus Logical NOT Bitwise NOT type caste indirection address size in byte	++,-- +,- ! ~ Type * & sizeof	Right to left
3	Multiplication Division modulus	* / %	Left to right
4	Addition Subtraction	+ -	Left to right
5	Left shift Right shift	<< >>	Left to right
6	Less than Less than equal to Greater than Greater than equal to	< <= > >=	Left to right
7	Equal to Not equal to	== !=	Left to right
8	Bitwise AND	&	Left to right
9	Bitwise XOR	^	Left to right
10	Bitwise OR	|	Left to right
11	Logical AND	&&	Left to right
12	Logical OR	||	Left to right
13	Conditional operator	?:	Right to left
14	Assignment operator or simple assignment Assignment by sum and difference Assignment by product and quotient Assignment by bitwise left shift and right shift Assignment by bitwise AND,XOR, and OR	= +=.-= *=,/=,%= <<=,>>= &=,^=,|=	Right to left
15	comma operator	,	Left to right

Difference between Unary & Binary Operators

Unary operator

The operator which operates on single operand known as Unary Operator.

Increment(++)
Decrement(--)
Address(&)
Unary minus(-)
One's complement(~)
Logical not (!)

Binary operator

The operators which operates on two operands known as Binary Operators. Some of the binary operators are:

Binary plus operator (+)
Binary minus operator (-)
Equals to (==)
Less than operator (<)
Grater than Operator (>)

Evaluation of Compound Expressions

Compound Epression: A compound expression is a series of simple expressions joined by arithmetic operators. A simple expression used in a compound expression must return a numeric value.

Program

```
#include<stdio.h>
int main ()
{
int a = 20, b = 10, c= 15, d = 5, e;
e = (a+b)*c/d;
printf("value of (a+b)*c/d is :%d\n",e);
e= ((a+b)*c)/d;
printf("value of ((a+b)*c)/d is : %d\n",e);
e = (a+b)*(c/d);
printf("value of (a+b)*(c/d) is: %d\n",e);
e = a+(b*c)/d;
printf("value of a+(b*c)/d is :%d\n",e);
reurn 0;
}
```

Output :

```
value of (a+b)*c/d is : 90
value of ((a+b)*c)/d is :90
value of (a+b)*(c/d) is:90
value of a+(b*c)/d is:50
```

Type Conversion Techniques

Type conversion in c is the process of converting one data type to another.
Type conversion is only performed to those data types where conversion is possible
Type conversion is performed by a compiler and it is done at compile time .
In type conversion the destination data type can't be smaller than the source data type.

There are two types of conversions:

1. **Implicit type conversion**
2. **Explicit type conversion**

Implicit type conversion

It is an automatic type conversion done by the compiler by it's own without any external trigger from the user.
It convert the lower data type to higher data type to avoid data loss.
It is also called type promotion(upgrade to largest data type).
Generally it takes place when in an expression more than one data type is present. In such conditions type conversion (type promotion) takes place to avoid loss of data.

```
bool -> char -> short int -> int ->
unsigned int -> long -> unsigned ->
long long -> float -> double -> long double
ex: short a = 20;
    int b = a;
```

Program

```
#include<stdio.h>
int  main ( )
{
int  x = 10;
char  y= 'a';
x = x+y;
float z = x+10;
printf("x = %d, z = %f",x,z);
return 0;
}
```

Output: x =107,z = 108.00000

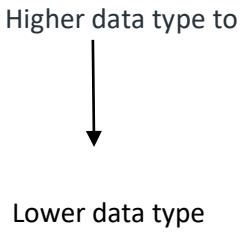
Explicit type Conversion

This process is also called type casting and it is user-defined. Here the user can typecast the result to make it of a particular data type.

This action is done forcefully and there is data loss. Some conversions cannot be made implicit like int to short.

The syntax in C Programming:

(type) expression



Program

```
#include<stdio.h>

int main ()

{

double x =1.2;
```

```
int sum = (int) x +1;

printf(“sum = %d”,sum);

Return 0;

}
```

Output :

Sum = 2