UNIT II

Bitwise Operators: Exact Size Integer Types, Logical Bitwise Operators, Shift Operators, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.

Selection & Making Decisions: Logical Data and Operators, Two Way Selection, Multiway Selection, More Standard Functions, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.

Repetition: Concept of Loop, Pretest and Post-test Loops, Initialization and Updating, Event and Counter Controlled Loops, Loops in C, Other Statements Related to Looping, Looping Applications, Programming Example The Calculator Program, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.

➢ Bitwise Operators:-

- These operators are used to perform bit operations. Decimal values are converted into binary values which are the sequence of bits and bit wise operators work on these bits.
- Bit wise operators in C language are & (bitwise AND), | (bitwise OR), ~ (bitwise NOT), ^ (XOR), << (left shift) and >> (right shift).

 TRUTH TABLE FOR BIT WISE OPERATION & BIT WISE OPERATORS

BELOW ARE THE BIT-WISE OPERATORS AND THEIR NAME IN C LANGUAGE.

1. & – Bitwise AND
2. | – Bitwise OR
3. ~ – Bitwise NOT
4. ^ – XOR
5. << – Left Shift
6. >> – Right Shift

➢ Exact Size Integer Types:-

- The following table provides the details of standard integer types with their storage sizes and value ranges −

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

➢ Bitwise operators are used for manipulating a data at the bit level, also called as bit level programming. Bit-level programming mainly consists of 0 and 1. They are used in numerical computations to make the calculation process faster.

Following is the list of bitwise operators provided by 'C' programming language:

Operators	Example/Description
&& (logical AND)	(x>5)&&(y<5) It returns true when both conditions are true
|| (logical OR)	(x>=10)||(y>=10) It returns true when at-least one of the condition is true
! (logical NOT)	!((x>5)&&(y<5)) It reverses the state of the operand "((x>5) && (y<5))" If "((x>5) && (y<5))" is true, logical NOT operator makes it false

➢ Shift Operators:-

Bitwise Right Shift Operator in C :-

1.It is denoted by **>>**

2. Bit Pattern of the data can be **shifted by specified number of Positions to Right.**

3. **When Data is Shifted Right , leading zero's are filled with zero**.

4. Right shift Operator is **Binary Operator** [Bi – two]

5. Binary means , **Operator that require two arguments**

- **Bitwise Left Shift Operator in C :-**

<< (left shift) Takes two numbers, left shifts the bits of the first operand, the second operand decides the number of places to shift. Or in other words left shifting an integer "x" with an integer "y" (x<<y) is equivalent to multiplying x with 2^y (2 raise to power y).

Important Points:-

- The left shift and right shift operators should not be used for negative numbers. The result of is undefined behaviour if any of the operands is a negative number. For example results of both -1 << 1 and 1 << -1 is undefined.
- If the number is shifted more than the size of integer, the behaviour is undefined. For example, 1 << 33 is undefined if integers are stored using 32 bits. See this for more details.
- The left-shift by 1 and right-shift by 1 are equivalent to multiplication and division by 2 respectively.

- **Selection & Making Decisions:-**
  - Decision making structures require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.
  - Show below is the general form of a typical decision making structure found in most of the programming languages –

- C programming language assumes any **non-zero** and **non-null** values as **true**, and if it is either **zero** or **null**, then it is assumed as **false** value.

1. If Statement

"If statement" is the selection statement used to select course of action depending on the conditions given. Therefore programmers can use this statement to control the flow of their program.

The syntax of the if statement in C programming is:

```
if (test expression)
{
   // statements to be executed if the test expression is true
}
```

2. C if...else Statement

The if statement may have an optional else block. The syntax of the if..else statement is:

```
if (test expression) {
    // statements to be executed if the test expression is true
}
else {
    // statements to be executed if the test expression is false
}
```

Nested if...else

It is possible to include an if..else statement inside the body of another if...else statement.

➢ Logical Data and Operators:-

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

Try the following example to understand all the logical operators available in C −

```
#include <stdio.h>

main() {

   int a = 5;
   int b = 20;
   int c ;

   if ( a && b ) {
      printf("Line 1 - Condition is true\n" );
   }

   if ( a || b ) {
      printf("Line 2 - Condition is true\n" );
   }

   /* lets change the value of  a and b */
   a = 0;
   b = 10;

   if ( a && b ) {
      printf("Line 3 - Condition is true\n" );
   } else {
      printf("Line 3 - Condition is not true\n" );
   }

   if ( !(a && b) ) {
      printf("Line 4 - Condition is true\n" );
   }

}
```

```
Line 1 - Condition is true
Line 2 - Condition is true
Line 3 - Condition is not true
Line 4 - Condition is true
```

When you compile and execute the above program, it produces the following result −

```
Line 1 - Condition is true
Line 2 - Condition is true
Line 3 - Condition is not true
Line 4 - Condition is true
```

- **Two Way Selection:-**
  - The two-way selection is the basic decision statement for computers.
  - The decision is based on resolving a binary expression, and then executing a set of commands depending on whether the response was true or false.
  - C, like most contemporary programming languages, implements two-way selection with the if…else statement.

- **Multi-Way Selection:-**

  A multi-way selection statement is used to execute **at most ONE** of the choices of a set of statements presented.

  Syntax of the multi-way select statement:

  Switch (Expression)
  {
  Case Constant1, one or more statements; break;
  Case constant2, one or more statements; break;
  ……..
  [default : one or more statements;]
  }

- **More Standard Functions:-**
  - The standard functions are built-in functions. In C programming language, the standard functions are declared in header files and defined in .dll files.
  - In simple words, the standard functions can be defined as "the ready made functions defined by the system to make coding more easy".
  - The standard functions are also called as **library functions** or **pre-defined functions**.
  - in C when we use standard functions, we must include the respective header file using **#include** statement.

- For example, the function **printf()** is defined in header file **stdio.h** (Standard Input Output header file).
- When we use **printf()** in our program, we must include **stdio.h** header file using **#include<stdio.h>** statement
- C Programming Language provides the following header files with standard functions.

<ctype.h> character testing and conversion functions.
<math.h> Mathematical functions
<stdio.h> standard I/O library functions
<stdlib.h> Utility functions such as string conversion routines memory allocation routines , random number generator,etc.
<string.h> string Manipulation functions
<time.h> Time Manipulation functions

MATH.H

abs : returns the absolute value of an integer x
cos : returns the cosine of x, where x is in radians
exp: returns "e" raised to a given power
fabs: returns the absolute value of a float x
log: returns the logarithm to base e
log10: returns the logarithm to base 10
pow : returns a given number raised to another number
sin : returns the sine of x, where x is in radians
sqrt : returns the square root of x

❖ Repetition:-

➢ Concept of Loop:-

- In looping, a program executes the sequence of statements many times until the stated condition becomes false. A loop consists of two parts, a body of a loop and a control statement.
- The control statement is a combination of some conditions that direct the body of the loop to execute until the specified condition

Types of Loops

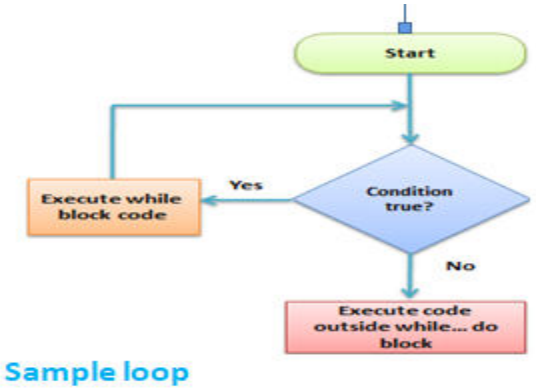
Depending upon the position of a control statement in a program, a loop is classified into two types:

1. Entry controlled loop

2. Exit controlled loop

In an **entry controlled loop,** a condition is checked before executing the body of a loop. It is also called as a pre-checking loop.

In an **exit controlled loop**, a condition is checked after executing the body of a loop. It is also called as a post-checking loop



Sample loop

The control conditions must be well defined and specified otherwise the loop will execute an infinite number of times.

'C' programming language provides us with three types of loop constructs:

1. The while loop

2. The do-while loop

3. The for loop

While Loop

A while loop is the most straightforward looping structure. The basic format of while loop is as follows:

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

It is an entry-controlled loop. In while loop, a condition is evaluated before processing a body of the loop.

Following program illustrates a while loop:

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

```
#include<conio.h>
int main()
{
        int num=1;      //initializing the variable
        while(num<=10)          //while loop with condition
        {
                printf("%d\n",num);
                num++;          //incrementing operation
        }
        return 0;
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Do-While loop

A do-while loop is similar to the while loop except that the condition is always executed after the body of a loop. It is also called an exit-controlled loop.

The basic format of while loop is as follows:

```
do {

 statements

} while (expression);
```

As we saw in a while loop, the body is executed if and only if the condition is true. In some cases, we have to execute a body of the loop at least once even if the condition is false. This type of operation can be achieved by using a do-while loop

In the do-while loop, the body of a loop is always executed at least once. After the body is executed, then it checks the condition. If the condition is true, then it will again execute the body of a loop otherwise control is transferred out of the loop.

The following program illustrates the working of a do-while loop:

```
#include<stdio.h>
#include<conio.h>
int main()
{
	int num=1;	//initializing the variable
	do	//do-while loop
	{
		printf("%d\n",2*num);
		num++;		//incrementing operation
	}while(num<=10);
	return 0;
}
```

Output:

```
2
4
6
8
10
12
14
16
18
20
```

For loop

A for loop is a more efficient loop structure in 'C' programming. The general structure of for loop is as follows:

```
for (initial value; condition; incrementation or decrementation )
{
  statements;
}
```

- The initial value of the for loop is performed only once.
- The condition is a Boolean expression that tests and compares the counter to a fixed value after each iteration, stopping the for loop when false is returned.
- The incrementation/decrementation increases (or decreases) the counter by a set value.

Following program illustrates the use of a simple for loop:

```
#include<stdio.h>
int main()
{
	int number;
	for(number=1;number<=10;number++) //for loop to print 1-10 numbers
	{
		printf("%d\n",number);		//to print the number
	}
	return 0;
```

}

Output:

1
2
3
4
5
6
7
8
9
10

Pretest and Post-test Loops:-

a) In a pretest loop, the test condition for the execution of the repetition statement is evaluated before executing each run of the loop.

In a posttest loop, the test condition for the execution of the repetition statement is evaluated after executing each run of the loop.

The differences between pretest and posttest loops are described in the table below:

S.No.	Pretest Loops	Posttest Loops
1.	Test condition is evaluated before executing any statement within the loop.	Test condition is evaluated after executing all statements within the loop.
2.	If the test condition is false, the loop statements never execute.	Even if the test condition is false, the loop statements execute once.
3.	The `for` and `while` statements are pretest loops in C++.	The `do-while` statement is posttest loop in C++.

Initialization:-

- Initialization is the process of locating and using the defined values for variable data that is used by a computer program. For example, an operating system or application program is installed with default or user-specified values that determine certain aspects of how the system or program is to function.
- The process of the user specifying initialization values is sometimes called *configuration*.

➢ Event and Counter Controlled Loops:-

The *Event-Controlled* while loop

- In this **while** loop an action is repeated until a certain event occurs. This is by far the most frequently used form of the **while** loop.
- There are three different types of **Event-Controlled** while loops

1. **Sentinel-controlled loops** - A *sentinel* variable is initialized to a specific value. The **while** loop continues until, through some action inside the loop, the *sentinel* variable is set to a predefined *termination* value.

2. **End-of-file controlled loops** - This type of **while** loop is usually used when reading from a file. The loop terminates when the end of the file is detected. This can easily be determined by checking the **EOF()** function after each read from the file. This function will return true when the end-of-file is reached.

3. **Flag controlled loops** - A **bool** variable is defined and initialized to serve as a *flag*. The **while** loop continues until the *flag* variable value flips (true becomes false or false becomes true).

➢ Count-Controlled Repetition:-

Count-controlled repetition requires

- control variable (or loop counter)
- initial value of the control variable
- increment (or decrement) by which the control variable is modified each iteration through the loop
- condition that tests for the final value of the control variable

- A count-controlled repetition will exit after running a certain number of times. The count is kept in a variable called an index or counter.
- When the index reaches a certain value (the loop bound) the loop will end.
- Count-controlled repetition is often called definite repetition because the number of repetitions is known before the loop begins executing.

➢ Other Statements Related to Looping:-

❖ break and continue:-

- ***break*** and ***continue*** are two C/C++ statements that allow us to further control flow within and out of loops.
- ***break*** causes execution to immediately jump out of the current loop, and proceed with the code following the loop.
- ***continue*** causes the remainder of the current iteration of the loop to be skipped, and for execution to recommence with the next iteration.

- In the case of ***for*** loops, the incrementation step will be executed next, followed by the condition test to start the next loop iteration.
- In the case of ***while*** and ***do-while*** loops, execution jumps to the next loop condition test.

❖ **Infinite Loops:-**

- Infinite loops are loops that repeat forever without stopping.
- Usually they are caused by some sort of error, such as the following example in which the wrong variable is incremented:

```
int i, j;
for( i = 0; i < 5; j++ )
    printf( "i = %d\n", i );
printf( "This line will never execute\n" );
```

❖ **Nested Loops**

The code inside a loop can be any valid C code, including other loops.

Any kind of loop can be nested inside of any other kind of loop.

➢ Looping Applications:-

Why do we use looping in programming?

Because we want to repeat something:

- Count from 1 to 10.
- Go through all the words in a dictionary to see whether they're palindromes.
- For each customer that has an outstanding balance, send out an email reminder that payment is due.
- For each directory under this one, find music files and add then to the list of known music.

➢ **The Calculator Program**:-

/*C program to design calculator with basic operations using switch.*/

```
#include <stdio.h>
 int main()
{
   int num1,num2;
   float result;
   char ch;    //to store operator choice
```

```
    printf("Enter first number: ");
   scanf("%d",&num1);
   printf("Enter second number: ");
   scanf("%d",&num2);
     printf("Choose operation to perform (+,-,*,/,%): ");
   scanf(" %c",&ch);
    result=0;
 switch(ch)
   {
      case '+':
         result=num1+num2;
         break;

      case '-':
         result=num1-num2;
         break;

      case '*':
         result=num1*num2;
         break;

      case '/':
         result=(float)num1/(float)num2;
         break;

      case '%':
         result=num1%num2;
         break;
      default:
         printf("Invalid operation.\n");
   }

   printf("Result: %d %c %d = %f\n",num1,ch,num2,result);
   return 0;
}
```

Output

```
Enter first number: 10
Enter second number: 20
Choose operation to perform (+,-,*,/,%): +
Result: 10 + 20 = 30.000000

Enter first number: 10
Enter second number: 3
Choose operation to perform (+,-,*,/,%): /
Result: 10 / 3 = 3.333333

Enter first number: 10
```

```
Enter second number: 3
Choose operation to perform (+,-,*,/,%): >
Invalid operation.
Result: 10 > 3 = 0.000000
```

The End