GBEST 204 Programming in C

Module 1

Part – II

C - Operators

C Fundamentals - Syllabus

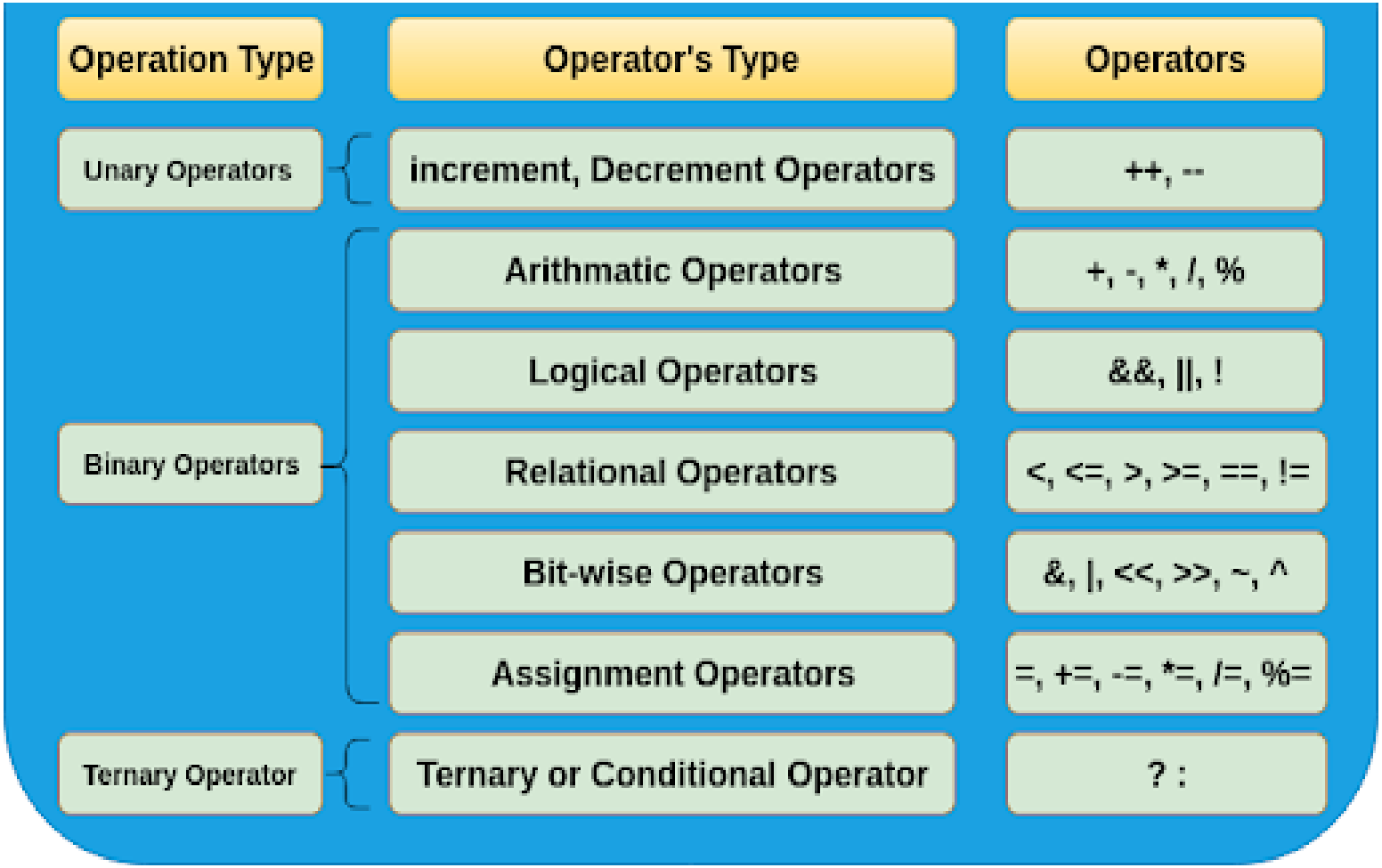
- Character Set, Constants, Identifiers, Keywords
- Basic Data types
- Variables
- **Operators and its precedence**
- **Bit-wise operators**
- **Expressions**
- Statements - Input and Output statements
- Structure of a C program
- Simple programs.

Operators

Unary operators: require one operand

Binary operators: require two operands

Ternary operators: require three operands



Unary operator

operator	operation	Initial value	Expression	Final value
+ +	Adds 1 to operand	x = 3	x++ ++x	x = 4
- -	Subtracts 1 from operand	y = 3	y-- --y	y = 2

Assignment operator

operator	operation	Initial value	Expression	Final value
a += 1	a = a+1	a = 6	a += 4	a = 10
b -= 1	b = b-1	b = 7	b -= 5	b = 2
c *= 1	c = c*1	c = 4	c *= 7	c = 28
d /= 2	d = d/2	d = 28	d/= 4	d = 7
a %= b	a= a%b	a = 55; b=2	a%=b	a = 1

Arithmetic Operators

They are used to perform various arithmetic operations

Operator	Meaning	Example	Result
+	Addition	6 + 3	9
		6.0 + 3.0	9.0
-	Subtraction	8 - 1	7
*	Multiplication	2 * 4	8
		2.0 * 4.0	8.0
/	Division	20/3	6
		20.0/3.0	6.6667
%	Modulus	5%3	2

Note: The ** is not a valid operator in C

Use of Arithmetic operator

1. WAP to convert Fahrenheit to Celsius

   Celsius = ((Fahren-32)*5)/9

2. WAP to convert days to months and days

Relational Operators

They are used to compare the values on either side of the operator

The result of a comparison is either **True** or **False**.

<	Less than	a<b
>	Greater than	a>b
<=	Less than or equal to	a<=b
>=	Greater than or equal to	a>=b
==	is equal to	a==b
!=	is not equal to	a!=b

Logical operators

Operator	Description
&&	Logical **AND** operator. If both the operands are non-zero, then the condition becomes true.
II	Logical **OR** Operator. If any of the two operands is non-zero, then the condition becomes true.
!	Logical **NOT** Operator. Used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.

Logical operators

Operator	Description	Example
&&	AND	x=6 y=3 x<10 && y>1 Return True
||	OR	x=6 y=3 x==5 || y==5 Return False
!	NOT	x=6 y=3 !(x==y) Return True

Use of Logical operator

```
#include<stdio.h>
int main()
{
int a;
printf("Enter any number : ");
scanf("%d", &a);
if(a%3 == 0 && a%5 == 0)
    printf("The given number %d is divisible by 3 and 5", a);
else
    printf("The given number %d is not divisible by 3 and 5", a);
return 0;
}
```

Output:

Enter any number : 45

The given number 45 is divisible by 3 and 5

WAP to check if number lies between 3 and 10

```
#include <stdio.h>
int main()
{
   int x = 5;
   printf("%d", x > 3 && x < 10);
   return 0;
}
```

Output:
1

// Returns 1 (True) because 5 is greater than 3 AND 5 is less than 10

Decimal – Binary conversion

Decimal	Binary	Decimal	Binary
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

Bitwise operators

- Bitwise operators take the binary representation of the operands and work on their bits, one bit at a time.
- The bits of the operand(s) are compared starting with the rightmost bit – LSB, then moving towards the MSB.

Bitwise operators

Operator	Description
& Binary AND	Result is 1 if both the corresponding operand bits are 1
| Binary OR	Result is 1 if atleast one of the corresponding operand bit is 1
^ Binary XOR	Result is 1 only if the corresponding bits do not match
~ Binary negation	It has the function of 'flipping' bits and is unary.
<< Binary Left Shift	The left operand's value is moved to its left by the number of bits specified in the right argument.
>> Binary Right Shift	The quantity of bits provided by the right parameter advances the position of the left operand.

Bitwise operators

Input		Output		
		AND	OR	XOR
bit1	bit2	bit1 & bit2	bit1 | bit2	bit1 ^ b2
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Bitwise operators

AND

```
    a = 0001 0100
    b = 0110 1100
        ---------
a & b = 0000 0100
      = 4
```

OR

```
    a = 0001 0100
    b = 0110 1100
        ---------
a | b = 0111 1100
      = 124
```

XOR

```
    a = 0001 0100
    b = 0110 1100
        ---------
a ^ b = 0111 1000
      = 120
```

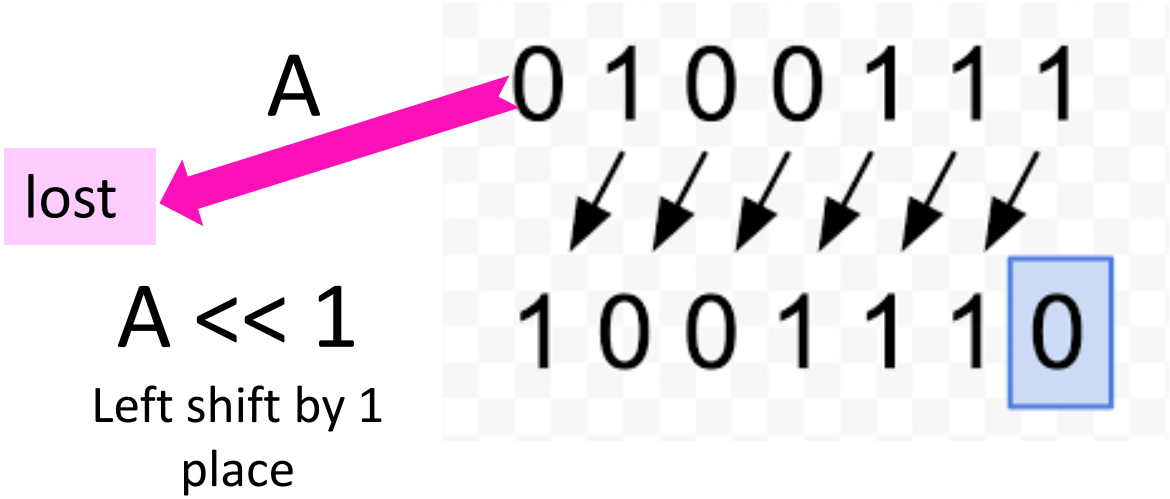
Negation

a = 100101

~a = 011010

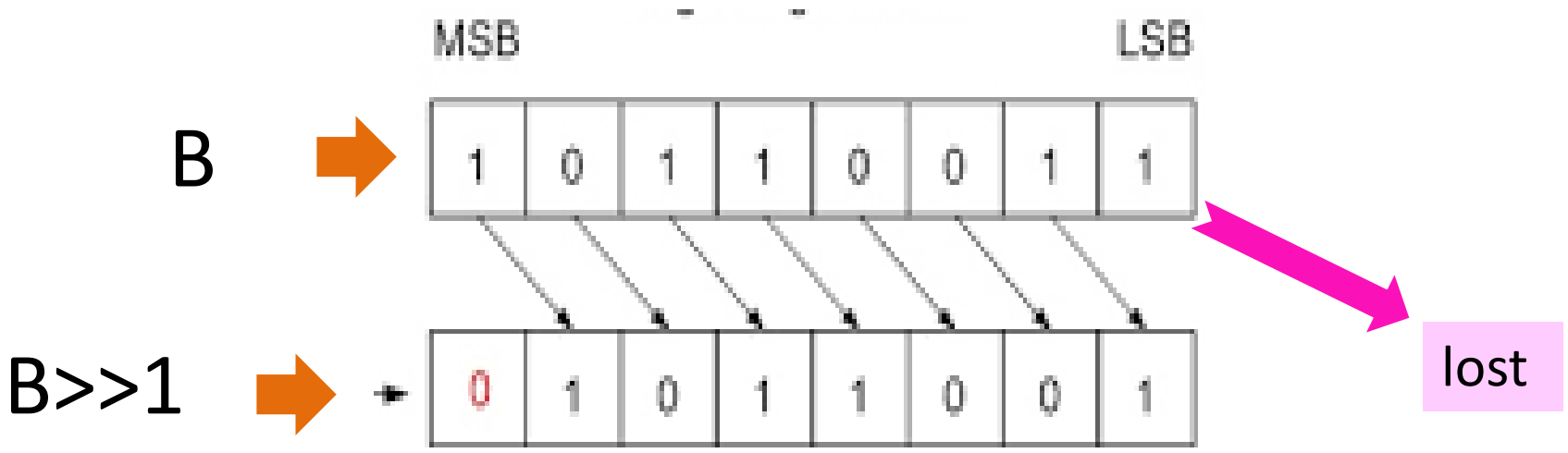
Bitwise left shift <<

- When we perform a left shift on binary number, we will move the bits to the left, adding a new bit to the right of the binary.
- This new bit will always be 0 for this operation, resulting in a new binary.
- The bit that falls of the end are lost



Bitwise right shift >>

- We will move the bits to the right. Thus, the bit farthest to the right of the old binary will be removed from the new binary.



Operator precedence

Operator precedence determines which operators have higher precedence than others.

Category	Operator	Associativity
Function call, Array	() []	Left to right
Unary	+ - ++ - - ! ~ & sizeof (type)	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right

Operator precedence

Category	Operator	Associativity
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	? :	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Table 3.8 *Summary of C Operators*

Operator	Description	Associativity	Rank
( ) [ ]	Function call Aray element reference	Left to right	1
+ − ++ −− ! ~ * & sizeof (type)	Unary plus Unary minus Increment Decrement Logical negation Ones complement Pointer reference (indirection) Address Size of an object Type cast (conversion)	Right to left	2
* / %	Multiplication Division Modulus	Left to right	3
+ −	Addition Subtraction	Left to right	4
<< >>	Left shift Right shift	Left to right	5
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	Left to right	6
== !=	Equality Inequality	Left to right	7
&	Bitwise AND	Left to right	8
^	Bitwise XOR	Left to right	9
|	Bitwise OR	Left to right	10
&&	Logical AND	Left to right	11
||	Logical OR	Left to right	12
?:	Conditional expression	Right to left	13
= * = /= %= += −= &= ^= |= <<= >>=	Assignment operators	Right to left	14
,	Comma operator	Left to right	15

Conditional operator ? :

Syntax: expression 1 ? expression 2 : expression 3

If expression 1 is true then

the value returned will be expression 2

otherwise the value returned will be expression 3".

Alternatively…

variable = condition ? valueIfTrue : valueIfFalse;

Conditional operator contd

`result = (age >= 18) ? "Adult" : "Minor";`	if age≥ 18 result = “Adult” else result = “Minor”
`max = (a > b) ? a : b;`	if a>b max = a else max = b

WAP to find if a number is odd or even

Using conditional operator

```
#include <stdio.h>
int main()
{
   int num;
   printf("Enter a number: ");
   scanf("%d", &num);
   (num % 2 ==0) ? printf("%d is even.\n", num)
                 : printf("%d is odd.\n", num);
   return 0;
}
```

Output:

Enter a number: 87

87 is odd.

Evaluate using conditional operator

WAP to evaluate the function using ternary operator

$$Y = \begin{cases} 1 & x > 5 \\ 0 & x < 5 \end{cases}$$

WAP to evaluate the function using ternary operator $Y = \begin{cases} 1 & x > 5 \\ 0 & x < 5 \end{cases}$

```
#include<stdio.h>
int main()
{
   int x,Y;
   printf("Enter any number : ");
   scanf("%d", &x);
   Y = (x > 5) ? 1 : 0;
   printf("The value of Y is : %d", Y);
   return 0;
}
```

Output 1:

Enter any number : -3

The value of Y is : 0

Output 2:

Enter any number : 56

The value of Y is : 1

WAP to find if given year is leap year

using conditional operator

```
#include <stdio.h>
int main() {
 int year;
 printf("Enter a year: ");
 scanf("%d", &year);
 ((year %4 ==0 && year %100 != 0) || (year %400 == 0))
        ? printf("%d is a leap year.\n", year)
        : printf("%d is not a leap year\n", year);
  return 0;
}
```

Output:

Enter a year: 2200

2200 is not a leap year.

sizeof - operator

```
#include <stdio.h>
int main()
{
   int a = 3;
   float b = 3.56;
   double c = 45.987654321;
   char d = 'Y';
   printf("Size of a is : %lu \n", sizeof(a));
   printf("Size of b is : %lu \n", sizeof(b));
   printf("Size of c is : %lu \n", sizeof(c));
   printf("Size of d is : %lu \n", sizeof(d));
   return 0;
}
```

It is used to compute the size of its operand.

It returns the size in number of bytes.

Output:

Size of a is : 4
Size of b is : 4
Size of c is : 8
Size of d is : 1

Division - datatype

Write a C program to divide two integer.

```c
#include<stdio.h>
int main()
{
 int a, b, c;
 a = 165;
 b = 100;
 c = a/b;
 printf("%d",c);
 return 0;
}
```

Output : 1

```c
#include<stdio.h>
int main()
{
 int a,b;
 float c;
 a = 165;
 b = 100;
 c = a/b;
 printf("%f",c);
 return 0;
}
```

Output : 1.000000

Division – datatype contd

Write a C program to divide two integer.

```
#include<stdio.h>
int main()
{
  int a;
   float b,c;
   a = 165;
   b = 100;
   c = a/b;
  printf("%f",c);
   return 0;
}
```

Output : 1.650000

```
#include<stdio.h>
int main()
{
 int a,b;
 float c;
 a = 8;
 b = 3;
 c = (float)a/b;
 printf("%f",c);
 return 0;
}
```

Output : 1.650000

(Type) operator for conversion

- float to int - causes truncation
- double to float – causes rounding of digits

Operator precedence
Home work

Suppose $a, b\ and\ c$ are integer variables that have been assigned the values $a = 8, b = 3\ and\ c = -5$. Determine the value of each of the following arithmetic expressions.

$a + b + c$	$a\%c$
$2 * b + 3 * (a - c)$	$a * b/c$
a/b	$a * (b/c)$
$a\%b$	$(a * c)\%b$
a/c	$a * (c\%b)$

Operator precedence
Home work

Suppose $x, y\ and\ z$ are floating point variables that have been assigned values $x = 8.8,\ y = 3.5\ and\ z = -5.2$ determine the value of each of the following arithmetic expressions.

1. $x + y + z$
2. $2 * y + 3 * (x - z)$
3. x/y
4. $x\%y$
5. $x/(y + z)$
6. $(x/y) + \mathrm{z}$
7. $2 * x/3 * y$
8. $2 * x/(3 * y)$