Computer system & Programming NOTES

UNIT - 2

Rajesh Tripathi
SIET

Introduction about C

C is a general-purpose high level language that was originally developed by Dennis Ritchie in AT & T BELL LAB in U.S.A for the Unix operating system. It was first implemented on the Digital Equipment Corporation PDP-11 computer in 1972. C has now become a widely used professional language for various reasons.

- Easy to learn
- Structured language
- It produces efficient programs.
- It can handle low-level activities.
- It can be compiled on a variety of computers.

Why to use C ?

C was initially used for system development work, in particular the programs that make-up the operating system. C was adoped as a system development language because it produces code that runs nearly as fast as code written in assembly language. Some examples of the use of C might be:

- Operating Systems
- Language Compilers
- Assemblers
- Text Editors
- Print Spoolers
- Network Drivers
- Modern Programs
- Data Bases
- Language Interpreters
- Utilities

C - Program Structure

A C program basically has the following form:

- Preprocessor Commands
- Functions
- Variables
- Statements & Expressions
- Comments

The following program is written in the C programming language. Open a text file ***hello.c*** using vi editor and put the following lines inside that file.

```
#include <stdio.h>

int main()
{
   /* My first program */
   printf("Hello, World! \n");

   return 0;
}
```

Preprocessor Commands: These commands tells the compiler to do preprocessing before doing actual compilation. Like *#include <stdio.h>* is a preprocessor command which tells a C compiler to include stdio.h file before going to actual compilation.

Functions: are main building blocks of any C Program. Every C Program will have one or more functions and there is one mandatory function which is called *main()* function. This function is prefixed with keyword *int* which means this function returns an integer value when it exits. This integer value is retured using *return* statement.

The C Programming language provides a set of built-in functions. In the above example *printf()* is a C built-in function which is used to print anything on the screen. Check Builtin function section for more detail.

Variables: are used to hold numbers, strings and complex data for manipulation.

Statements & Expressions : Expressions combine variables and constants to create new values. Statements are expressions, assignments, function calls, or control flow statements which make up C programs.

Comments: are used to give additional useful information inside a C Program. All the comments will be put inside /*...*/ as given in the example above. A comment can span through multiple lines.

Note the followings

- C is a case sensitive programming language. It means in C *printf* and *Printf* will have different meanings.
- C has a free-form line structure. End of each C statement must be marked with a semicolon.
- Multiple statements can be one the same line.
- White Spaces (ie tab space and space bar) are ignored.
- Statements can continue over multiple lines.

C Compilers

When you write any program in C language then to run that program you need to compile that program using a C Compiler which converts your program into a language understandable by a computer. This is called machine language (ie. binary format). So before proceeding, make sure you have C Compiler available at your computer.

Q. What is C Token? Explain types of C token used in C language.

- **Ans.** C tokens are the basic buildings blocks in C language which are constructed together to write a C program.
- Each and every smallest individual units in a C program are known as C tokens.

- C tokens are of six types. They are,

1. Keywords (eg: int, while),
2. Identifiers (eg: main, total),
3. Constants (eg: 10, 20),
4. Operators (eg: +, /,-,*)

1. **Keyword**: The following names are reserved by the C language. Their meaning is already defined, and they cannot be re-defined to mean anything else.

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

IDENTIFIER

- Each program elements in a C program are given a name called identifiers.
- Names given to identify Variables, functions and arrays are examples for identifiers. eg. x is a name given to integer variable in above program.

The Programming language C has two main variable types

- Local Variables
- Global Variables

Identifier rules:

1. Variable can single character or combination of character or digits.
2. Each variable have the first character must be alphabets.
3. No any special character required within the variable names.
4. Only underscore(_) sign can be used within the variable names.
5. No any keyword can be used as a variable names.

Constants:

The term constant means that it does not change its value during the execution of program. In the language C, constant and is the data with a constant value that does not change in the program. For example, in the program "100" "3.14" "'A'" "" Hello "" and the like, if you write data directly, and constant. Moreover, also called a literal constant. Constant expression is an expression consisting only of constants. There are four basic types of constants in C. They are:

1. Integer constants
2. Floating-point constants
3. Character constants
4. String constants

Integer constants

Integer constants are whole numbers without any fractional part. Thus integer constants consist of a sequence of digits. Integer constants can be written in three different number systems: Decimal, Octal and Hexadecimal.

A decimal integer constant consists of any combination of digits taken from the set 0 through 9. If the decimal constant contains two or more digits, the first digit must be something other than 0. The following are valid decimal integer constants.

0 1 1234 -786

Floating-point constants

A floating-point constant can be written in two forms: Factorial form or Exponential form. A floating-point constant in a fractional form must have at least one digit each to the left and right of the decimal point. A floating-point in exponent form consists of a mantissa and an exponent.
The following are valid floating-point constants.</1.0 0.1 2E-4 -0.1555e-4

Character constants

A character constant is a single character, enclosed in single quotation marks.
e.g., 'A' 'B' '1'
Character constants are usually just the character enclosed in single quotes; 'a', 'b', 'c'. Some characters can't be represented in this way, so we use a 2 character sequence as follows.

'\n'	newline
'\t'	tab
'\\'	backslash
'\"'	single quote
'\0'	null (Usedautomatically to terminate character string)

String constants

A string constant consists of zero or more character enclosed in quotation marks. Several string constants are given below.

"Welcome to C Programming"

Q. What is Operator? Explain types of C operator used in C Language?

ANS: Simple answer can be given using expression *4 + 5 is equal to 9*. Here 4 and 5 are called operands and + is called operator. C language supports following type of operators.

- Arithmetic Operators
- Logical (or Relational) Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

Arithmetic Operators:

There are following arithmetic operators supported by C language:

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide numerator by denumerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

Logical Relational) Operators: There are following logical operators supported by C language
Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
= =	Checks if the value of two operands is equal or not, if yes then condition becomes true.	(A = = B) is not true.
!=	Checks if the value of two operands is equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Relational Operators:

&&	Called Logical AND operator. If both the operands are non zero then then condition becomes true.	(A && B) is true.
||	Called Logical OR Operator. If any of the two operands is non zero then then condition becomes true..	(A || B) is true
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is false.

Bitwise Operators:
Bitwise operator works on bits and perform bit by bit operation.
Assume if A = 60; and B = 13; Now in binary format they will be as follows:
A = 0011 1100
B = 0000 1101

A&B = 0000 1100
A|B = 0011 1101
A^B = 0011 0001
~A = 1100 0011
There are following Bitwise operators supported by C language

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in eather operand.	(A | B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the efect of 'flipping' bits.	(~A) will give -60 which is 1100 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111

Assignment Operators: There are following assignment operators supported by C language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assigne value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It	C /= A is equivalent to C = C / A

	divides left operand with the right operand and assign the result to left operand	
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

Short Notes on L-VALUE and R-VALUE:
x = 1; takes the value on the right (e.g. 1) and puts it in the memory referenced by x. Here x and 1 are known as L-VALUES and R-VALUES respectively L-values can be on either side of the assignment operator where as R-values only appear on the right.
So x is an L-value because it can appear on the left as we've just seen, or on the right like this: y = x; However, constants like 1 are R-values because 1 could appear on the right, but 1 = x; is invalid.
Misc Operators: There are few other operators supported by C Language.
Show Examples

Operator	Description	Example
sizeof()	Returns the size of an variable.	sizeof(a), where a is interger, will return 4.
&	Returns the address of an variable.	&a; will give actaul address of the variable.
*	Pointer to a variable.	*a; will pointer to a variable.
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

Operators Categories:
All the operators we have discussed above can be categorized into following categories:

- Postfix operators, which follow a single operand.
- Unary prefix operators, which precede a single operand.
- Binary operators, which take two operands and perform a variety of arithmetic and logical operations.
- The conditional operator (a ternary operator), which takes three operands and evaluates either the second or third expression, depending on the evaluation of the first expression.
- Assignment operators, which assign a value to a variable.
- The comma operator, which guarantees left-to-right evaluation of comma-separated expressions.

Precedence of C Operators:
Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator:
For example x = 7 + 3 * 2; Here x is assigned 13, not 20 because operator * has higher precedenace than + so it first get multiplied with 3*2 and then adds into 7.
Here operators with the highest precedence appear at the top of the table; those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

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

Q. What is data type? Explain types of data type used in C language?

Data Type in C

C has a concept of 'data types' which are used to define a variable before its use. The definition of a variable will assign storage for the variable and define the type of data that will be held in the location. Data type is used to determine what type of value a variable or a constant can contain throughout the program. In C language, different variables contain different data types

Th In C language, it is compulsory to declare variables with their data type before using them in any statement. Mainly data types are categorized into 3 categories:-

1. Fundamental Data Types
2. Derived Data Types
3. User Defined Data Types

1. Fundamental Data Types

Fundamental data types are further categorized into 3 types. Let us discuss each of this type briefly.

Int (Integer)

Integer data type is used to store numeric values without any decimal point e.g. 7, -101, 107, etc.
A variable declared as 'int' must contain whole numbers e.g. age is always in number.

Syntax:
int variable name;
E.g. int roll, marks, age;

Float

Float data type is used to store numeric values with decimal point. In other words, float data type is used to store real values, e.g. 3.14, 7.67 etc. A variable declared as float must contain decimal values e.g. percentage, price, pi, area etc. may contain real values.

Syntax:

Float variable name;

E.g. float per, area;

Char (Character)

Char (Character) data type is used to store single character, within single quotes e.g. 'a', 'z','e' etc. A variable declared as 'char' can store only single character e.g. Yes or No Choice requires only 'y' or 'n' as an answer.

Syntax:

Char variable name;

E.g. Char chi='a', cha;

Void

Void data type is used to represent an empty value. It is used as a return type if a function does not return any value.

Data Type-name	**Type**	**Range**
Int	Numeric – Integer	-32 768 to 32 767
short	Numeric – Integer	-32 768 to 32 767
long	Numeric – Integer	-2 147 483 648 to 2 147 483 647
float	Numeric – Real	1.2×10^{-38} to 3.4×10^{38}
double	Numeric – Real	2.2×10^{-308} to 1.8×10^{308}
char	Character	All ASCII characters

2. Derived data types:-

i. **Array**
ii. **Structure**
iii. **Pointer**
iv. **Union**
v. **function**

i. Array:-An array type describes a contiguously allocated nonempty set of objects with a particular member object type, called the element type. Array types are characterized by their element type and by the number of elements in the array. An array type is said to be derived from its element type, and if its element type is T, the array type is sometimes called *array of T*. The construction of an array type from an element type is called *array type derivation*.
ii. Structure:-A structure type describes a sequentially allocated nonempty set of member objects (and, in certain circumstances, an incomplete array), each of which has an optionally specified name and possibly distinct type.
iii. Pointer:- A pointer type may be derived from a function type, an object type, or an incomplete type, called the referenced type. A pointer type describes an object whose value provides a reference to an entity of the referenced type. A pointer type derived from the referenced type T is sometimes called *pointer to T*. The construction of a pointer type from a referenced type is called *pointer type derivation*.

iv. Union:- A union type describes an overlapping nonempty set of member objects, each of which has an optionally specified name and possibly distinct type.
v. Function:-A function type describes a function with specified return type. A function type is characterized by its return type and the number and types of its parameters. A function type is said to be derived from its return type, and if its return type is T , the function type is sometimes called *function returning T*. The construction of a function type from a return type is called *function type derivation*.

Data type modifiers in C

In c language Data Type Modifiers are keywords used to change the properties of current properties of data type. Data type modifiers are classified into following types.

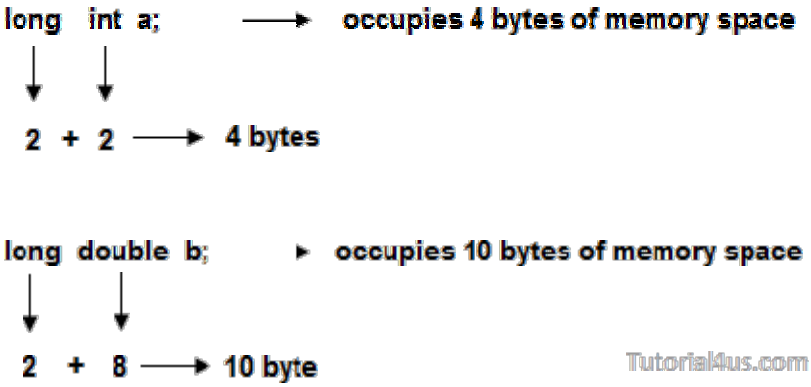
- long
- short
- unsigned
- signed

Modifiers are prefixed with basic data types to modify (either increase or decrease) the amount of storage space allocated to a variable.

For example, storage space for int data type is 4 byte for 32 bit processor. We can increase the range by using long int which is 8 byte. We can decrease the range by using short int which is 2 byte.

long:

This can be used to increased size of the current data type to 2 more bytes, which can be applied on int or double data types. For example int occupy 2 byte of memory if we use long with integer variable then it occupy 4 byte of memory.



Syntax

```
long a;  --> by default which represent long int.
```

short

In general int data type occupies different memory spaces for a different operating system; to allocate fixed memory space short keyword can be used.

Syntax

```
short int a; --> occupies 2 bytes of memory space in every operating system.
```

unsigned

This keyword can be used to make the accepting values of a data type is positive data type.

Syntax

```
unsigned int a =100;          // right
unsigned int a=-100;          // wrong
```

Signed

This keyword accepts both negative or positive value and this is default properties or data type modifiers for every data type.

Example

```
int a=10;             // right
int a=-10;            // right
signed int a=10;      // right
signed int a=-10;     // right
```

Note: in real time no need to write signed keyword explicitly for any data type.

Qualifiers

A type qualifier is used to refine the declaration of a variable, a function, and parameters, by specifying whether:

- The value of a variable can be changed.
- The value of a variable must always be read from memory rather than from a register

Standard C language recognizes the following two qualifiers:

- const
- volatile

The *const* qualifier is used to tell C that the variable value can not change after initialization.

const float pi=3.14159; Now *pi* cannot be changed at a later time within the program.

The **volatile qualifier** declares a data type that can have its value changed in ways outside the control or detection of the compiler (such as a variable updated by the system clock or by another program). This prevents the compiler from optimizing code referring to the object by storing the object's value in a register and re-reading it from there, rather than from memory, where it may have changed. You will use this qualifier once you will become expert in "C".

Q. What is INPUT and OUTPUT statement in C? Explain with syntax and Example.

ANS.Input and Output Statements

Reading, processing, and printing of data are the three essential functions of a computer program.
There are two methods of providing data to the program variables. One method is to assign values to variables through the assignment statements. Another method is to use input functions, which can get data from the keyboard (standard input-stdin).
There are two types of Input and Output (I/O) statements: Unformatted I/O statements and Formatted I/O statements.

Unformatted Input statements

Character Input

There are several functions available to input a character from the console.

getchar ()

This function accepts a single character from the stream stdin (keyboard buffer). This single character includes alphabets, digits, punctuations, return, and tab.

General form:

char-variable = getchar();
getch (); - character input from console & doesn't echo the character.
getche(); - character input from console & echoes the character.

getch():

getch() is a nonstandard function and is present in conio.h header file which is mostly used by MS-DOS compilers like Turbo C. Like above functions, it reads also a single character from keyboard. But it does not use any buffer, so the entered character is immediately returned without waiting for the enter key.

getche Function: The getche() = get character echo. It is also used to get a single character from the keyboard during the program execution. When this function is executed, the character entered by the user is displayed on the screen.

Syntax: **Variable name=getche();**

gets (): This function accepts a string terminated by a new line character. Blank space is also considered

as a character. To get a line of text, this function serves the purpose.
General Form:
gets(stringvariable); /* string is represented as character array */
Example
char ch[5];
gets(ch);

Unformatted Output statements
Character Output:
putchar():-This function displays a single character in the standard output (stdout), monitor.

General Form:
putchar(char variable);
char ch;
ch = getchar();
putchar(ch);
String Output
puts():-This function displays the string in the standard output.
General Form:
puts(str);

Example char ch[5];
gets(ch);
puts(ch);

Formatted I/O Statements
Formatted input refers to an input data that has been arranged in a particular format. C has a special formatting character (%). A character following this defines the format for a value. Some of the format specifies are given below:

%c – character
%d – integer
%f, %e, %g – float
%s – string
%ld – long integer
%o – octal
%lf—double
%Lf—long double
%x – hexadecimal
%hd – short integer
%[..] – string of specified characters
%u – unsigned

Formatted Input Statement
scanf():- scanf () function is used to read formatted data items.
General Form:
scanf ("format string", list of variables);
Format string specifies the field format in which the data is to be entered. List of variables specify the address of memory locations where the data is to be stored. Address operator (&) is used before the variables. Format string and variables are separated by comma. Format string, also known as control string contains field specifications, which directs the interpretation of input data. By default, the delimiter
while reading the values is space. Delimiter can be user-defined. To read a string using '%s', '&'
need not be used.

Example
scanf ("%c %d %f", &ch, &i, &x);
scanf ("%[^\n]s", str);
scanf ("%d=%d", &a, &b);
scanf ("%2d%5d",&a,&b);

scanf (“%d%d”, &a,&b);
sscanf()
sscanf() function to read values from a string. This functions returns the number of inputs read successfully.
General Form:
sscanf (str, “format string”, list of variables);

Formatted Output Statement
printf():-printf () function is used to output the values. This function returns the number of characters printed.

General Form:
printf (“format string”, list of variables);

Example
printf (“char=%c, int=%3d, floating point=%6.2f”,ch, i, x);
printf (“sum = %*.*f”, w, p, sum);
printf (“name = %10.4s”, name);

sprintf():- sprintf() function is used to output values to a string.
General Form:
sprintf (str, “format string”, list of variables);

Summary
_ C is a structured programming language.
_ C program is a collection of functions.
_ C supports four basic primitive data types: int, char, float, double.
_ C has a rich set of operators.
_ C has Unformatted and Formatted Input / Output statements.

Q. What is storage class? Explain types of storage class used in C language.

Ans. storage class defines the scope (visibility) and life time of variables and/or functions within a C Program. These specifiers precede the type that they modify. There are following storage classes which can be used in a C Program

- auto
- register
- static
- extern

The auto Storage Class
The **auto** storage class is the default storage class for all local variables.

```
{
  int mount;
  auto int month;
}
```

The example above defines two variables with the same storage class, auto can only be used within functions, i.e. local variables.

The register Storage Class

The **register** storage class is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and can't have the unary '&' operator applied to it (as it does not have a memory location).

```
{
  register int  miles;
}
```

The register should only be used for variables that require quick access such as counters. It should also be noted that defining 'register' goes not mean that the variable will be stored in a register. It means that it MIGHT be stored in a register depending on hardware and implementation restrictions.

The static Storage Class

The **static** storage class instructs the compiler to keep a local variable in existence during the lifetime of the program instead of creating and destroying it each time it comes into and goes out of scope. Therefore, making local variables static allows them to maintain their values between function calls.

The static modifier may also be applied to global variables. When this is done, it causes that variable's scope to be restricted to the file in which it is declared.

In C programming, when **static** is used on a class data member, it causes only one copy of that member to be shared by all objects of its class.

```
#include <stdio.h>

/* function declaration */
void func(void);

static int count = 5; /* global variable */

main()
{
  while(count--)
  {
    func();
  }
  return 0;
}
/* function definition */
void func( void )
{
  static int i = 5; /* local static variable */
  i++;

  printf("i is %d and count is %d\n", i, count);
}
```

You may not understand this example at this time because I have used *function* and *global variables* which I have not explained so far. So for now let us proceed even if you do not understand it completely. When the above code is compiled and executed, it produces following result:

```
i is 6 and count is 4
i is 7 and count is 3
i is 8 and count is 2
i is 9 and count is 1
i is 10 and count is 0
```

The extern Storage Class: The extern storage class is used to give a reference of a global variable that is visible to ALL the program files. When you use 'extern' the variable cannot be initialized as all it does is point the variable name at a storage location that has been previously defined.

When you have multiple files and you define a global variable or function which will be used in other files also, then *extern* will be used in another file to give reference of defined variable or function. Just for understanding *extern* is used to declare a global variable or function in another files.

The extern modifier is most commonly used when there are two or more files sharing the same global variables or functions as explained below.

Second File: write.c

```
#include <stdio.h>

extern int count;

void write_extern(void)
{   count = 5;
   printf("count is %d\n", count);
}
```

Here *extern* keyword is being used to declare *count* in the second file where as it has its definition in the first file. Now compile these two files as follows: Decision making structures require that the programmer specify 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.