Chapter 1

Programming in C

LEARNING OBJECTIVES

- *Basic concepts*
- *Character set*
- *Identifier*
- *Declaring a variable*
- *Visualization of declaration*
- *Constants*
- *Single character constants*
- *String constants*
- *Using const keyword*
- *Precedence decreases as we move from top to bottom*
- *Type conversion*
- *Documentation section*
- *Preprocessing*
- *Global declaration*
- *Control statements*
- *Selection/Decision making statement*
- *Looping statements*
- *Unconditional jump statements*

Basic Concepts

Character Set

A character refers to an alphabet, digit or a special symbol. Alphabets: $A-Z$, $a-z$

Digits: 0 -9
Special symbols:
~ ! # % ∧ and ∗ ( ) − + { } [ ] − < > , . | ? \ | : ; ” ’ White space

Identifier

Identifier is a user-defined name used for naming a variable or a function.
Rules for naming an identifier

- Consists only letters, digits and underscore
- Starts only with an alphabet or underscore
- Keywords cannot be used.
- Can be as long as you like, first 31 characters are significant.

Example: Valid identifiers: RollNo, Roll_No, _Roll_No
rollno, Name2;
Invalid: 2name, Roll No.

Variable

The name itself represents value, is not constant. Variable is a data name whose value varies/changes during program execution. Variable name is a name given to memory cell (may be one or multiple bytes).

Data Types

Represents type of data and set of operations to perform on data .

Data Type

Primitive/Basic	Derived	User defined	Valueless
– Char	– Array	– Structure	
– float	– pointer	– union	– void
– double		Enumeration	
– integer			

Type	Keyword	Number of Bytes
Integer	int	2
Floating	float	4
Double	double	8
Character	char	1

Declaring a Variable

- Before using a variable, you must give some information to compiler about the variable. i.e., you must declare it.
- Declaration statement includes the type and variable name.

Syntax:
Datatype Var_name;
Example:

```
int roll_no;
char ch;
float age;
```

- When we declare a variable
  - memory space is allocated to hold a value of specified type.
  - space is associated with variable name
  - space is associated with a unique Address.

Table 1 *Visualization of declaration*

	roll no
int roll no;	$\boxed{\text{garbage}}$
	2002
	marks
int marks = 10;	$\boxed{10}$
	3008
	diameter
float diameter = 5.9	$\boxed{5.9}$
	4252
	ch → variable name
char ch : 'A'	$\boxed{A}$ → value
	2820 → address

Note: The default value is garbage, i.e., an unknown value is assigned randomly.

Renaming data types with typedef Typedef is a keyword, which can form complex types from the basic type, and will assign some simpler names for such combinations. This is more helpful when some declaration is very tough, confusing or varies from one implementation to another.

For example, the data type unsigned long int is redefined as LONG as follows:

typedef unsigned long int LONG;

Uses of enumerated data types Enumerated data types are most useful when one is working over small, discrete set of values, in which each is having a meaning and it is not a number.

A best example can be given on months jan, feb, mar, …, dec, which are 12 in number, with assigning consecutive numbers for it.

The main advantages are storage efficiency, the *c*-code can become readable

Constants

A constant value is one which does not change during the execution of a program.
C supports several types of constants:

1. Integer constants
2. Real constants
3. Single character constants
4. Strings constants

Integer constants

An integer constant is a sequence of digits. It consists of a set of digits 0 to 9 preceded by an optional + or − sign spaces, commas, and non-digit characters are not permitted between digits.
Examples for valid decimal integer constants are
123
–31
0
562321
+78
Examples for invalid integer constants are
20,000
₹1000

Real constants

Real constants consist of a fractional part in their representation. Integer constants are inadequate to represent quantities that vary continuously.
Examples of real constants are
0.0026
–0.97
435.29
+487.0

Single character constants

A single character constant represents a single character which is enclosed in a pair of quotation symbols.
Examples for character constants are
'5'
'x'
';'

String constants

A string constant is a set of characters enclosed in double quotation marks. The characters in a string constant sequence may be alphabet, number, special character and blank space.
Examples of string constants are
"VISHAL"
"1234"
"C language"
"!….?"

Naming constants

A name given to a constant value. Value of name does not change during program execution.

Using const keyword

When we use 'const' with data type, memory will be allocated to variable and the initialized value does not change.
const int $x = 10$;
const float pi = 3.141;

Using # define

# define x 10
# define pi 3.141
Where '# define' is instruction to preprocessor so memory is allocated. The preprocessor replace each occurrence of name with value in program before execution.

Operator

An operator is a symbol which performs operations on given data elements.

Table 2 *Precedence and Associativity*

( ) Parenthesis [ ] Index → Member of • Member of	$L-R$
Pre ++, − − (unary) − , &(address of) * (Indirection)	$R-L$
Arithmetic * , /, %	$L-R$
Arithmetic: +, −	$L-R$
Bitwise shift : ≪, ≫	$L-R$
Relational: <, >, =. >, > = = =, ! =	$L-R$
Bitwise ex −OR : ∧	$L-R$
Logical AND : &&	$L-R$
Logical OR : ||	$L-R$
Conditional: ? :	$R-L$
Assignment & compound Assignment =, + =, − =, * =, / = ; % =	$R-L$
Separation operator: , (comma)	$L-R$

Note: For Assignment operator, only a variable is allowed on its left.

Precedence Decreases as We Move from Top to Bottom

Examples:

1. int a,b,c;
   $a = b = c = 0$;
   Assigns '0' to a,b,c;
2. int a, $b = 55$, $c = 10$;
   initializes 'b' with 55 and 'c' with '10'.
   $b + c = a$; // Invalid
   Only variable is allowed on left side of assignment.
3. int $a = 15$, $b = 20$, $c = 2$, $d = 5$, $e = 10, f, g, h, i$;
   $f = a << c$;
   'a' is left shifted for 'c' times and result stored in 'f'
   i.e.,
   $a = 15 = (1\ 1\ 1\ 1)_2$
   ↓ ↓ ↓ ↓
   1 1 1 1 0 (After first shift)
   ↓ ↓ ↓ ↓
   1 1 1 1 0 0 (After second shift)
   One left shift multiplies 15 by 2 = 30
   Again the 2nd left shift multiplies 30 by 2 = 60
   Thus $15 \times 2^2 = 60$, where the power of 2 is the number of times shift is made. Value of 'f' becomes 60.

Note: Left shift multiplies the value by 2. Right shift divides the value by 2.

$g = a$ and b;

$a - 0\ 1\ 1\ 1\ 1$
$b - 1\ 0\ 1\ 0\ 0$

$0\ 0\ 1\ 0\ 0 = 4$

'&' performs bitwise AND. So 'g' value is '4'.

$h = a|b$; $a - 0\ 1\ 1\ 1\ 1$
$b - 1\ 0\ 0\ 0\ 0$

$1\ 1\ 1\ 1\ 1 = 31$

"I" performs bitwise 'OR'. R value is '31'.

$i = a\,\hat{}\,d$: $a - 1\ 1\ 1\ 1$
$b - 0\ 1\ 0\ 1$

$1\ 0\ 1\ 0 = 10$

'^' performs bit-wise ex – OR. i value is '10'.

4. int $a = 100$, $b = 200$, $c = 300, x$;
```
x = (a>b)?((a>c)?a:c):((b>c)? b:c);
      false                      c
x = c
so, x = 300
```
5. int $i = 10, j = 10, x, y$;
```
  x = i+++++i+i+++++i+++i
  executes as
  ++ i⎫
  ++ i⎬pre-increments
  ++ i⎭
  X = i + i + i + i + i
  i ++ ;⎫post-increments
  i ++ ;⎭
  so x = 65, i = 15.
  y = j - - + - - j + j - -  + - - j +
  - - j
```

```
executes as
- - j;⎫
- - j;⎬pre decrements
- - j;⎭
y = j + j + j + j + j ;
j - -;⎫
j - -;⎭post decrements.
y = 35; j = 5
```

6. int *i* = 10;

```
printf("%d%d%d%d%d", i++, ++i, ++i,
i++, ++i);
evaluates the values in printf from
right to left.
So
i++, ++i, ++i, i++, ++i
<-----------------------
Prints 14 14 13 11 11
Printf ("%d", i)
Prints 15:
```

Type Conversion

'*C*' allows mixed mode operations, i.e., variables of different type may appear in same expression. To perform the operation, the data need to convert into compatible type.

The conversion takes place in two ways:

Implicit

C automatically converts any intermediate values to proper type so that the expression can be evaluated without losing any significance.

For mixed mode operations, generally the 'lower' type is automatically converted to 'higher' type before the operation proceeds.

Explicit

'*C*' allows programmer to use type conversion operator to convert a data value to the required type.

Syntax:
*V*1 = (type) *V*2;
Type in parenthesis represents the destination type.

Example: int *a* = 3, *b* = 2, float *x*, *y*;

Case I: *x* = *a*/*b*;
results *x* = 1.000000

Case II: *y* = (float) *a*/*b*;
results *y* = 1.500000.

Because, in case 1, the integer division is performed and so returns an integer by division operator. While assigning the integer value implicitly converted to 1.000000, then assigns to float variable *x* where as in case 2, (float)a converts value of '*a*' to float, the second variable '*b*' is integer. The compiler implicitly converts integer to float. Then it performs float division. So 1.500000 is stored into floating variables.

Notes: '*C*' allows both implicit and explicit type conversion. Type conversion is of two types:

1. Narrowing**:** Conversion of 'higher' type to 'lower' type.
2. Widening**:** Conversion of 'lower' type to 'higher' type.

Widening
$\longrightarrow$
Char – int – long – float – double – long double
$\longleftarrow$
Narrowing

Note: Narrowing causes loss of data.

Input/output Functions

Function	Purpose
printf	prints formatted string
scanf	reads formatted string
getchar	reads character
putchar	displays a character
gets	reads a string
puts	displays a string

Format Specifier	Purpose
%c	single character
%d	decimal integer
%e	floating point
%f	floating point
%h	short int
%o	octal integer
%x	hexa decimal
%s	string
%u	unsigned decimal integer

Note: scanf("%s", string_var); does not read string which contains white space. Hence to read multi word string use gets(string_var);

Example 1: Which of following comment regarding the reading of a string using scanf( ) and gets ( ) is true?
(A) Both can be used interchangeably
(B) scanf is delimited by end of line, gets is delimited by blank space
(C) scanf is delimited by blank, gets is delimited by end of line
(D) None of these

Ans: (C)

Program Structure

```
/* Documentation section */
Preprocessor commands;
Global declaration;
main ()
```

```
{
Body of main;
}
User defined function area;
```

Documentation section/comments Ignored by compiler, provides additional information to user to improve readability.

Preprocessing Tells the compiler to do pre-processing before doing compilation. For example

#include < stdio.h > tells to include stdio header file.

Global declaration It contains variable declarations, these are accessible in more than one function.

Function Functions are main building blocks of 'C' program. Every 'C' program contains one or more functions. A mandatory function called 'main( )' instructs the compiler to start execution from here.

User defined area Here user can define his own functions.

Control Statements

The statement that controls the execution sequence of a program is called "control statement".

The control statements are classified as:

1. Selection statement: if, switch
2. Iterative/looping statement: While, do-while, for
3. Unconditional jump statements: break, continue, return, goto

Selection/Decision-making Statement

Makes a decision to select and execute statement(*s*) based on the condition. '*C*' supports if and switch selection statements.

The if statement "if" is called two-way selection statement.

Syntax:

```
if (expression) // simple-if
       statement(s);
if (expression) // if-else
{
       statement1(s);
}
else
{
       statements(s);
}
if (expression) // ladder else-if.
{
       Statement1(s);
}
else if (expression2)
{
Statement2(s);
}
else
{
Statement3(s);
}
```

Nested if:

```
       if (expression1)
{
       Statement(s)1;
       if (expression(s)2)
 else
       Statement(s)3;
}
else
Statement(s)4;
```

Note: If the expression evaluates to true then the statements of if block gets executed otherwise else block statements will execute.

Example 2: Consider the following program segment:

```
if (a > b) printf ("a > b");
else
printf ("else part");
printf ("a < = b");
a < = b will be printed if
```

(A) $a > b$ (B) $a < b$
(C) $a = b$ (D) all of these

Ans: (D)

Because the statement, printf("$a < = b$"); is not the part of either if block or else block.

The switch statement Switch is a multi-way (*n*-way) selection statement.

Syntax:

```
switch (var_name/exp)
{
case const1: stmts1;
             break;
case const2: stmts2;
             break;
         .
         .
         .
case constn: stmts n;
             break;
default: statements;
}
```

Notes:

- For switch only the integral (integer/char) type variables or expression evaluates to integral allowed.
- Absence of break after case statements leads continuation execution of all case statements followed by matching case block.

Example 3:

```
main ()
{
int i = 10;
switch(i)
{
case 10 : printf ("case 10");
case 15 : printf ("case 15");
case 20 : printf ("case 20");
default : printf ("default case");
}
}
```

Output: Case 10 case 15 case 20 default case

Reason: Missing break after each case, leads to execution of all the cases from matching case.

Example 4:

```
main ( )
{
int i = 10;
switch (i)
{
case 10 : printf("case 10");
break ;
case 8 + 2 : printf("case 8+2");
            break;
default : printf(" No matching case");
}
}
```

Program raises an error called 'Duplicate case' while compiling because the expression '8 + 2' evaluates to '10'.

Looping Statements

Sometimes, there is a situation to execute statement(s) repeatedly for a number of times or until the condition satisfies. *C*' supports following looping statements: while, do-while, for.

While Statement

Syntax: while (condition)

```
{
Statement(s);
}
```

If the condition is true the block of statements will execute and control returns to condition, i.e., the statement(*s*) executes till the condition becomes false.

Notes:

- 'While' executes the block either '0' or more times.
- 'While' is called entry control loop.

Do-while Statement.

Syntax:

```
do
          {
                 Statement(s);
          } while (condition);
```

do-while is same as 'while; except that the statement(s) will execute for at least once.

Notes:

- The condition will not be evaluate to execute the block for first time.
- 'do-while' is called exit-control loop.

Example 5:

```
main ( )
{
int i = 0;
while (i! = 0)
{
        printf("%d", i);
        i++;
}
}
```

No output, because the condition is false for the first time.

```
main ( )
        {
        int  i = 0;
        do
        {
        printf("%d", i);
        i++;
} while (i! = 0);
}
```

Output: Displays 0 to 32767 and–32768 to–1

The for loop 'for' provides more concise loop control structure.

Syntax:

```
for(exp1; exp2; exp3)
{
Statement(s);
}
```

Expression 1: Initialization expression may contain multiple initializations. It executes only once before executing the loop for first time.

Expression 2: Condition expression. Only one condition expression is allowed. That may be single or compound condition, evaluates before every execution.

Expression 3: Modification statement may contain multiple statements. It executes on completion of loop body for every iteration.

Note: All the expressions in parenthesis are optional. Two semi-colons (;) are compulsory even though there are no expressions.

Odd loops In the for loop, while loop, the condition specifies the number of times a loop can be executed. Sometimes a user may not know, about the number of times a loop is to be executed. If we want to execute a loop for unknown number of times, then the concept of odd loops should be implemented, these can be done using the for, while (or) do-while loops. Let us illustrate odd–loop with a program

```
# include <stdio.h>
main()
{
int num, x;
num = 1;
while (num = = 1)
{
printf ("enter a number");
scanf ("%d", & x);
if((x % 2) = = 0)
printf("number is even");
else
printf("number is odd");
printf("do u want to test any num.");
printf("for yes-enter '1', No-enter '0'");
Scanf("%d",& num);
}
}
```

Unconditional Jump Statements

- "*C*" language permits to jump from one statement to another.
- '*C*' supports break, continue, return and goto jump statements.

Break statement Breaks the execution sequence. That is when the break statement executes in a block (loop) it'll come out from block (loop).

Syntax:

```
break;
```

Continue statement Used to skip a part of the loop under certain conditions.

Syntax:

```
continue;
```

Return statement Terminates the execution of a function and returns the control to the calling function.

Syntax:

```
return [exp/value];
```

Goto statement Jumps from one point to another with in a function.

Syntax:

```
        label1:                 goto label2:
        Statement(s);           Statement(s);
        goto label1;            label2;
        reverse jump            forward jump
```

Reverse jump, executes the statements repeatedly where as in forward jump, the statements are skipped from execution.

Example 6:

```
        main( )
        {
        int i ;
        for (i=1; i<=10; i++)
        {
                if (i = = 5)
                        break;
                printf("%d" , i);
        }
        }
```

output: 1 2 3 4
if *i* = 5, then the loop will break.

Example 7:

```
        main( )
        {
                int i ;
                for (i  = 1; i<=10; i++)
                {
                        if (i = = 5)
                        continue;
                        printf("%d" , i);
                }
        }
```

o/p: 1 2 3 4 6 7 8 9 10
if *i* = 5, the loop statements skipped for that iteration. So it does not print '5'.

Example 8:
Output for the following program segment

```
for (i = 1, j = 10 ; i < 6; ++i, --j)
printf("\n %d %d", i, j);
```

Output:

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

Note: Since for statement allows multiple initialization and multiple update statements, expression 1 and expression 3, does not raise any error.

Exercises

Practice Problems I

Directions for questions 1 to 15: Select the correct alternative from the given choices.

1. What will be the output of the following program?
```
void main()
{
int i;
char a[ ] =" \0 ";
if (printf("%s\n", a))
printf ("ok \n");
else
printf("program error \n");
}
```
(A) ok (B) progam error
(C) no output (D) compilation error

2. Output of the following will be
```
# define FALSE-1
# define TRUE 1
# define NULL 0
main( )
{
    if(NULL)
    puts("NULL");
    else if(FALSE)
    puts("TRUE");
    else
    puts("FALSE");
    }
```
(A) NULL (B) TRUE
(C) FALSE (D) 1

3. main( )
```
{
    printf("%x",-1 << 4) ;
}
```
For the above program output will be
(A) FFF0 (B) FF00
(C) 00FF (D) 0FFF

4. For the following program
# define sqr (*a*) *a***a*
main( )
```
    {
    int i;
    i = 64 / sqr(4);
    printf( "%d", i);
    }
```
output will be
(A) 4 (B) 16
(C) 64 (D) compilation error

5. #define clrscr ( ) 1000
```
main ( )
{
clrscr();
    printf ( "%d \n", clrscr());
   }
```
Output of the above program will be?
(A) error (B) No output
(C) 1000 (D) 1

6. Output of the following program is
```
main( )
{
    int i = -2;
    +i;
    printf("i = %d, +i = %d\n", i, +i);
```
(A) error (B) –2, +2
(C) –2, –2 (D) –2, 2

7. main( )
```
{
  int n;
  printf("%d", scanf ("%d", & n));
}
```
For the above program if input is given as 20. What will be the output?
(A) 20 (B) 1
(C) 2 (D) 0

8. How many times will the following code be executed?
```
{
    x = 10;
    while (x = 1)
            x ++;
}
```
(A) Never
(B) Once
(C) 15 times
(D) Infinite number of times

9. The following statement
`printf("%d", 9%5); prints`
(A) 1.8 (B) 1.0
(C) 4 (D) 2

10. int *a*;
`printf("%d", a);`
What is the output of the above code fragment?
(A) 0 (B) 2
(C) Garbage value (D) 3

11. printf("%d", printf("time"));
(A) syntax error
(B) outputs time 4
(C) outputs garbage
(D) prints time and terminates abruptly

12. The following program
```
main( )
{
    int i = 2;
    {
    int i = 4, j = 5;
```

```
        printf ("%d%d",i,j);
        }
        printf ("%d%d",i,j);
    }
```

(A) Compiler error: unrecognised symbol *j*;
(B) Prints 2545
(C) Print 4525
(D) None of the above

13. What is the output of the following program fragment?

```
for (i = 3; i < 15; i + = 3);
printf ("%d", i);
```

(A) a syntax error (B) an execution error
(C) prints 12 (D) prints 15

14. What is the output of the following program segment?

```
int a = 4, b = 6;
printf("%d", a = b);
```

(A) Outputs an error message
(B) Prints 0
(C) Prints 1
(D) None of these

15. The statements:

```
a = 7;
printf("%d", (a++));
prints
```

(A) Value of 8 (B) Value of 7
(C) Value of 0 (D) None of the above

Practice Problems 2

Directions for questions 1 to 12: Select the correct alternative from the given choices.

1. If the condition is missing in a FOR loop of a C program then
(A) It is assumed to be present and taken to be false
(B) It is assumed to be present and taken to be true
(C) It results in syntax error
(D) Execution will be terminated abruptly

2. Which of the following operators in '*C*' does not associate from the right?
(A) = (B) + =
(C) postfix++ (D) >

3. In a C programming language $x - = y + 1$ means
(A) $x = -x - y - 1$ (B) $x = x - y + 1$
(C) $x = x - y - 1$ (D) $x = -x + y + 1$

4. Minimum number of temporary variables needed to swap two variables is
(A) 1 (B) 2
(C) 3 (D) 0

5. A preprocessor command
(A) need not start on a new line
(B) need not start on the first column
(C) has # as the first character
(D) comes after the first executable statement

6. printf ("%d", printf ("%d", printf("time4kids")));
(A) Outputs time (B) Syntax error
(C) Outputs 9 (D) None of the above

7. for (i = 1; i < 5; i++)
if (i!=3)
printf("%d", i);
Outputs:
(A) 12345 (B) Error
(C) 1245 (D) 0000

8. Which operand in '*C*' takes only integer operands?
(A) * (B) /
(C) % (D) +

9. An unrestricted use of 'goto' statement is harmful because
(A) it results in increasing the executing time of the program
(B) it increases the memory of the program
(C) it decreases the readability and testing of program
(D) None of the above

10. What will be the output?

```
main()
{
int i = 0, j = 0;
if(i && j ++)
printf("%d..%d", i++, j);
printf("%d..%d", i, j);
}
```

(A) 1..1 (B) 2..2
(C) 0..0 (D) 1..1, 1..1

11. What is the output?

```
main ()
{
int a = 0;
int b = 20;
char x = 1;
char y = 10;
if(a, b, x, y);
printf("hello");
}
```

(A) logical error (B) Garbage value
(C) hello (D) 20

12. What will be the value of count after executing the below program:

```
main ( ) {
int count = 10, digit = 0;
while (digit < = 9) {
printf ("%d\n", ++count);
++digit;
}
}
```

(A) 10 (B) 11
(C) 20 (D) 21

Previous Years' Questions

1. Which one of the following are essential features of an object-oriented programming language?
   (i) Abstraction and encapsulation
   (ii) Strictly-typedness
   (iii) Type-safe property coupled with sub-type rule
   (iv) Polymorphism in the presence of inheritance

   [2005]

   (A) (i) and (ii) only
   (B) (i) and (iv) only
   (C) (i), (ii) and (iv) only
   (D) (i), (iii) and (iv) only

2. Which of the following are true?

   (i) A programming language which does not permit global variables of any kind and has no nesting of procedures/functions, but permits recursion can be implemented with static storage allocation

   (ii) Multi-level access link (or display) arrangement is needed to arrange activation records only if the programming language being implemented has nesting of procedures/functions

   (iii) Recursion in programming languages cannot be implemented with dynamic storage allocation

   (iv) Nesting procedures/functions and recursion require a dynamic heap allocation scheme and cannot be implemented with a stack-based allocation scheme for activation records

   (v) Programming languages which permit a function to return a function as its result cannot be implemented with a stack-based storage allocation scheme for activation records

   [2008]

   (A) (ii) and (v) only (B) (i), (iii) and (iv) only
   (C) (i), (ii) and (v) only (D) (ii), (iii) and (v) only

3. What will be the output of the following C program segment?
   char inChar = 'A';
   switch(inChar) {
   case 'A': printf("choice A\n"):
   case 'B':
   case 'C': printf("choice B");
   case 'D'
   case 'E':
   default: printf("No Choice");} **[2012]**
   (A) No choice
   (B) Choice A
   (C) Choice A
   Choice B No choice
   (D) Program gives no output as it is erroneous

4. Suppose n and p are unsigned int variables in a C program. We wish to set p to ${}^nC_3$. If n is large, which one of the following statements is most likely to set p correctly?

   [2014]

   (A) $p = n * (n - 1) * (n - 2)/6;$
   (B) $p = n * (n - 1) /2* (n - 2)/3;$
   (C) $p = n * (n - 1) /3 * (n - 2)/2;$
   (D) $p = n * (n - 1) * (n - 2)/6.0;$

5. The secant method is used to find the root of an equation $f(x) = 0$. It is started from two distinct estimates x_a and x_b for the root. It is an iterative procedure involving linear interpolation to a root. The iteration stops if $f(x_b)$ is very small and then x_b is the solution. The procedure is given below. Observe that there is an expression which is missing and is marked by ?. Which is the suitable expression that is to put in place of ? so that it follows all steps of the secant method?

   [2015]

   Secant

```
Initialize: $x_a$, $x_b$, $\varepsilon$, $N$   // $\varepsilon$ = convergence indicator
                                 // $N$ = maximum no. of
iterations
$f_b = f(x_b)$
$i = 0$
while ($i < N$ and $|f_b| > \varepsilon$) do
$i = i + 1$        // update counter
$x_t = ?$          // missing expression for
                   // intermediate value
$x_a = x_b$        // reset $x_a$
$x_b = x_t$        // reset $x_b$
$f_b = f(x_b)$     // function value at new $x_b$
end while
if $|f_b| > \varepsilon$ then            // loop is terminated with $i = N$
write "Non-convergence"
else
write "return $x_b$"
end if
```

   (A) $x_b - (f_b - f(x_a))\, f_b / (x_b - x_a)$
   (B) $x_a - (f_a - f(x_a))\, f_a / (x_b - x_a)$
   (C) $x_b - (x_b - x_a)\, f_b / (f_b - f(x_a))$
   (D) $x_a - (x_b - x_a)\, f_a / (f_b - f(x_a))$

6. Consider the following C program:

```
#include<stdio.h>
int main( )
{
int i, j, k = 0;
j = 2 * 3 / 4 + 2.0 / 5 + 8 / 5;
k -= --j;
for (i = 0; i < 5; i ++)
{
switch(i + k)
{
case 1:
case 2: printf("\n%d", i + k);
case 3: printf("\n%d", i + k);
default: printf(("\n%d", i + k);
}
}
return 0;
}
```

The number of times printf statement is executed is _______. **[2015]**

7. Consider the C program fragment below which is meant to divide x by y using repeated subtractions. The variables x, y, q and r are all unsigned int.

```
while (r >= y) {
    r = r – y;
    q = q + 1;
}
```

Which of the following conditions on the variables x, y, q and r before the execution of the fragment will ensure that the loop terminates in a state satisfying the condition x == (y*q + r)? **[2017]**

(A) (q == r) && (r == 0)
(B) (x > 0) && (r == x) &&(y > 0)
(C) (q == 0) && (r == x) && (y > 0)
(D) (q == 0) && (y > 0)

8. Consider the following C Program.

```
#include<stdio.h>
int main () {
        int m = 10;
        int n, nl ;
        n = ++m;
        nl = m++;
        n--;
        --nl;
        n  -= nl;
        printf ("%d",  n) ;
        return 0;
}
```

The output of the program is ___________. **[2017]**

Answer Keys

Exercises

Practice Problems 1

1. A	**2.** B	**3.** A	**4.** C	**5.** C	**6.** C	**7.** B	**8.** D	**9.** C	**10.** C
11. B	**12.** A	**13.** D	**14.** D	**15.** B					

Practice Problems 2

1. B	**2.** D	**3.** C	**4.** D	**5.** C	**6.** D	**7.** C	**8.** C	**9.** C	**10.** C
11. C	**12.** C								

Previous Years' Questions

1. B	**2.** D	**3.** C	**4.** B	**5.** C	**6.** 10	**7.** C	**8.** 0