Biyani's
Group of Girls' Colleges

B.C.A. I YEAR

Principles of Programming Language(Through C)

PRE-UNIVERSITY EXAM 2018(Set 2)

MAX MARKS :100

PART - I

A- Attempt all the questions (very short answer). Your answers should not exceed the maximum word limit of 40 for each question. Each question carries 2 marks.

1. What is algorithm?
   an **algorithm** is a finite sequence of instructions, a logic and explicit step-by-step procedure for solving a problem starting from a known beginning. – the number of instructions must be finite.

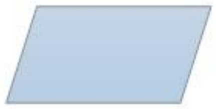
2. What is printf?

   printf() functions are inbuilt library functions in C programming language which are available in C library by default. These functions are declared and related macros are defined in "stdio.h" which is a header file in C language.
   printf() function is used to print the "character, string, float, integer, octal and hexadecimal values" onto the output screen.
   Example

```
printf("Integer value is %d\n" , no);
```

3. Give flow chart symbols for i/o, processing and flow line?

   Flowcharts use special shapes to represent different types of actions or steps in a process.
   Input/Output Symbol
   Represents material or information entering or leaving the system, such as customer order (input) or a product (output).



Action or Process Symbol

A box can represent a single step ("add two cups of flour"), or and entire sub-process ("make bread") within a larger process.

4. What is constant in C ?

Constantsrefer to fixed values that the program may not alter during its execution. These fixed values are also called literals. **Constants** can be of any of the basic data types like an integer **constant**, a floating **constant**, a character **constant**, or a string literal.

5. What do you understand by while(1234) ?

while loop in C programming repeatedly executes a target statement as long as a given condition is true.
Here condition is a non zero value so it is always true. Hence it is an infinite loop.

6. How do you read and write string in C?

One possible way to read in a string is by using *scanf*. However, the problem with this, is that if you were to enter a string which contains one or more spaces, scanf would finish reading when it reaches a space, or if return is pressed. As a result, the string would get cut off. So we could use the *gets* function

A *gets* takes just one argument - a char pointer, or the name of a char array, but don't forget to declare the array / pointer variable first! What's more, is that it automatically prints out a newline character, making the output a little neater.

A *puts* function is similar to *gets* function in the way that it takes one argument - a char pointer. This also automatically adds a newline character after printing out the string. Sometimes this can be a disadvantage, so *printf* could be used instead.

7. What is pointer ?.

A **pointer** is a variable which contains the address in memory of another variable. We can have a **pointer** to any variable type. The unary or monadic operator & gives the ``address of a variable''. The indirection or dereference operator * gives the ``contents of an object pointed to by a**pointer**''.

8. What is CONIO.H?

conio.h is a **C** header file used mostly by MS-DOS compilers to provide console input/output. It is not part of the **C** standard library or ISO **C**, nor is it defined by POSIX. This header declares several useful library functions for performing "console input and output" from a program.

9. What is header file?

A **header file** is a **file** containing **C** declarations and macro definitions (see Macros) to be shared between several source **files**. You request the use of a **header file** in your program by including it, with the **C** preprocessing directive ' #include '.

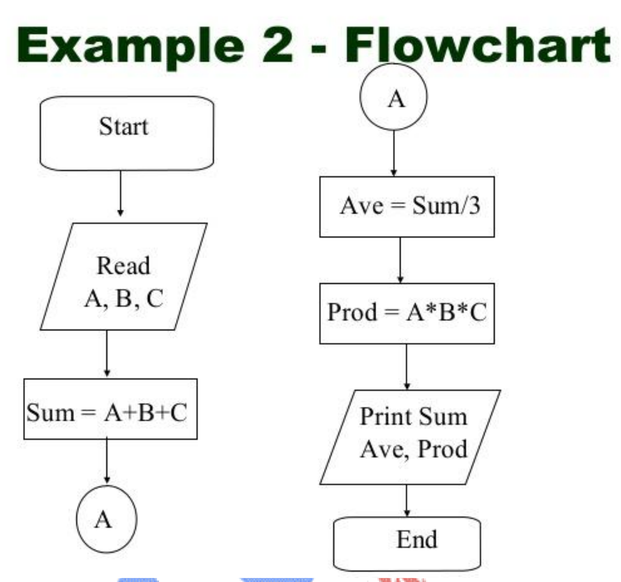
10. What is function definition?

A **function definition** specifies the name of the **function**, the types and number of parameters it expects to receive, and its return type. A **function definition** also includes a **function** body with the declarations of its local variables, and the statements that determine what the **function** does.

PART- II

B- Attempt all the questions (short answers). Your answer should not exceed the maximum word limit of 80 words for each question. Each question carriers 4 marks.

1. Draw a flow chart to find out sum and average of 3 numbers.



2. Explain data types in c with suitable example?
   Data types in c refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

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	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

3. Differentiate between break and continue statement with suitable example?

Difference Between break and continue

Difference Between break and continue

break	**continue**
A break can appear in both switch and loop (for, while, do) statements.	A continue can appear only in loop (for, while, do) statements.
A break causes the switch or loop statements to terminate the moment it is executed. Loop or switch ends abruptly when break is encountered.	A continue doesn't terminate the loop, it causes the loop to go to the next iteration. All iterations of the loop are executed even if continue is encountered. The continue statement is used to skip statements in the loop that appear after the continue.
The break statement can be used in both switch and loop statements.	The continue statement can appear only in loops. You will get an error if this appears in switch statement.
When a break statement is encountered, it terminates the block and gets the control out of the switch or loop.	When a continue statement is encountered, it gets the control to the next iteration of the loop.
A break causes the innermost enclosing loop or switch to be exited immediately.	A continue inside a loop nested within a switch causes the next loop iteration.

4. Explain various types of function?

   A function is a group of statements that together perform a task. Every C program has at least one function, which is **main()**, and all the most trivial programs can define additional functions.

   Types of functions:

C – Library functions

- Library functions in C language are inbuilt functions which are grouped together and placed in a common place called library.
- Each library function in C performs specific operation.
- We can make use of these library functions to get the pre-defined output instead of writing our own code to get those outputs.
- These library functions are created by the persons who designed and created C compilers.
- All C standard library functions are declared in many header files which are saved as file_name.h.
- Actually, function declaration, definition for macros are given in all header files.
- We are including these header files in our C program using “#include<file_name.h>” command to make use of the functions those are declared in the header files.

User defined functions

A **function** is a block of code that performs a specific task. **C**allows you to **define functions** according to your need. These **functions** are known as **user-defined functions** User defined functions are the functions which are written by us for our own requirement.

5. What is the purpose of getch() and scanf().

   getch**()** reads a single byte character from input. getch**()** is a way to get a user inputted character. It can be used to hold program execution, but the "holding" is simply a side-effect of its primary purpose, which is to wait until the user enters a character.
   scanf is a function that reads data with specified format from a given string stream source, originated from **C** programming language, and is present in many other programming languages. The **scanf** function prototype is: int **scanf**(const char *format, ...);

PART- III

C- Attempt all the questions (long answer). Draw neat and comprehensive sketches wherever necessary to clearly illustrate your answer. Each question carries 12 marks.

1. What is Programming languages? Explain each of them in detail.
   A programming language is a formal language that specifies a set of instructions that can be used to produce various kinds of output. Programming languages generally consist of instructions for a computer. Programming languages can be used to create programs that implement specific algorithms.
   Types of Languages
   There are three main kinds of programming language:

- Machine language
- Assembly language
- High-level language

We just went over what **machine language** is - it's the language of machines, consisting of bits (1s and 0s) put together into chunks like **bytes**, a group of 8 bits, and lots of other larger sizes. It's highly unlikely you will ever have to write in machine language, but in the old days, we used to plot 1s and 0s on graph paper and then type them in, to make pictures appear on the computer screen. Very tedious!

Assembly language is a little easier than machine language, but not much! It uses more convenient numbers, symbols, and abbreviations to describe the huge strings of 1s and 0s, to make it both easier and more memorable to type in instructions. The computer knows that certain strings of numbers are commands, so assembly language lets you use English-like

High level languages are the more common programming languages on the forum, C# or Python for example. These high level languages usually compile into lower level languages or machine code. Think of high level languages of someone basically making functions from lower level languages that can perform multiple things as one function. The programming and understanding will usually be easier in high level languages in addition to usually being faster, however high level languages can sometimes lead to slower or code with a lot of extra luggage that is not needed which is why you may use lower level languages if you are trying to make something as efficent as possible perhaps due to limitations. Not saying that it is not possible to make something just as efficient in a high level language but it can vary. High level languages may also take care of things like garbage collection automatically.

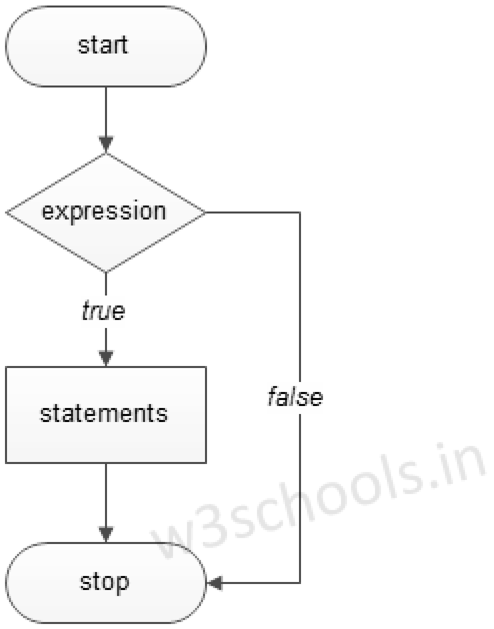
2. With the help of suitable example explain decision making statement of "C" in brief.

C conditional statements allow you to make a decision, based upon the result of a condition. These statements are called as Decision Making Statements or Conditional Statements.

So far, we have seen that all set of statements in a C program gets executed sequentially in the order in which they are written and appear. This occurs when there is no jump based statements or repetitions of certain calculations. But a number of situations may arise where we may have to change the order of execution of statements depending on some specific conditions. This involves a kind of decision making from a set of calculations. It is to be noted that C language assumes any non-zero or non-null value as true and if zero or null, treated as false.

This type of structure requires that the programmers indicate several conditions for evaluation within a program. The statement(s) will get executed only if the condition becomes true and optionally, alternative statement or set of statements will get executed if the condition becomes false.

<u>The flowchart of Decision-making technique in C can be expressed as:</u>



C languages have such decision-making capabilities within its program by the use of following the decision making statements:

Conditional Statements in C

<u>**if statement :**</u>

If statements in C is used to control the program flow based on some condition, it's used to execute some statement code block if the expression is evaluated to true, otherwise, it will get skipped. This is the simplest way to modify the control flow of the program.

```
if(test_expression) { statement 1; statement 2; ... }
```

if-else statement

If else statements in C is also used to control the program flow based on some condition, only the difference is: it's used to execute some statement code block if the expression is evaluated to true, otherwise executes else statement code block.

```
if(test_expression) { //execute your code } else { //execute your code }
```

Example :

```
#include<stdio.h>


main()

{

  int a, b;


  printf("Please enter the value for a:");

  scanf("%d", & amp; a);


  printf("\nPlease the value for b:");

  scanf("%d", & amp; b);


  if (a & gt; b) {

    printf("\n a is greater");

  } else {

    printf("\n b is greater");

  }

}
```

Nested if-else statement

Nested if else statements play an important role in C programming, it means you can use conditional statements inside another conditional statement.

```
if(test_expression one)
{
  if(test_expression two) {
    //Statement block Executes when the boolean test expression two is true.
  }
}
else
{
   //else statement block
}
```

else if-statement :

else-if statements in C is like another if condition, it's used in a program when if statement having multiple decisions.

```
if(test_expression) { //execute your code } else if(test_expression n) { //execute your code } else {
//execute your code }
```

goto statement :

C supports a special form of a statement that is the goto Statement which is used to branch unconditionally within a program from one point to another. Although it is not a good habit to use goto statement in C, there may be some situations where the use of goto statement might be desirable.

goto statement is used by programmers to change the sequence of execution of a C program by shifting the control to a different part of the same program.

```
goto label;
```

switch statement

C switch statement is used when you have multiple possibilities for the if statement.

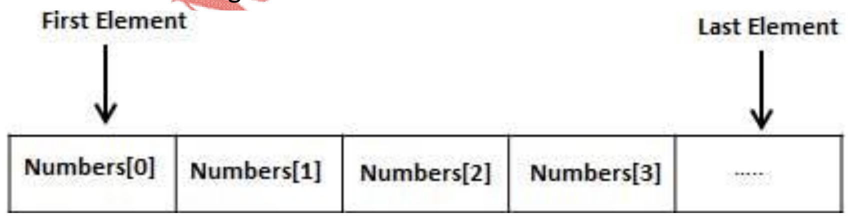
```
switch(variable)
{
```

```
case 1:
  //execute your code
break;

case n:
  //execute your code
break;

default:
  //execute your code
break;
}
```

After the end of each block it is necessary to insert a break statement because if the programmers do not use the break statement, all consecutive blocks of codes will get executed from each and every case onwards after matching the case block.

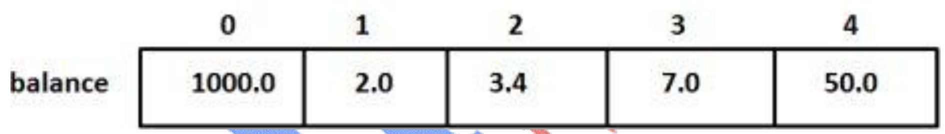
3. What is array? Find the sum and average of 100 numbers using Array.

   rrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
   Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.
   All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



Declaring Arrays

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows –
type arrayName [arraySize];
This is called a *single-dimensional* array. The **arraySize** must be an integer constant greater than zero and **type** can be any valid C data type. For example, to declare a 10-element array called **balance** of type double, use this statement –
double balance[10];
Here *balance* is a variable array which is sufficient to hold up to 10 double numbers.
Initializing Arrays
You can initialize an array in C either one by one or using a single statement as follows –
double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};
The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ].
If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write –
double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};
You will create exactly the same array as you did in the previous example. Following is an example to assign a single element of the array –
balance[4] = 50.0;
The above statement assigns the 5th element in the array with a value of 50.0. All arrays have 0 as the index of their first element which is also called the base index and the last index of an array will be total size of the array minus 1. Shown below is the pictorial representation of the array we discussed above –



Accessing Array Elements
An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example –
double salary = balance[9];
The above statement will take the 10th element from the array and assign the value to salary variable. The following example Shows how to use all the three above mentioned concepts viz. declaration, assignment, and accessing arrays –

```
#include <stdio.h>

int main () {

   int n[ 10 ]; /* n is an array of 10 integers */
   int i,j;

   /* initialize elements of array n to 0 */
   for ( i = 0; i < 10; i++ ) {
      n[ i ] = i + 100; /* set element at location i to i + 100 */
```

```
  }

  /* output each array element's value */
  for (j = 0; j < 10; j++ ) {
    printf("Element[%d] = %d\n", j, n[j] );
  }

  return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109
```

```
#include <stdio.h>

int main()
{
   int Arr[100], n, i, sum = 0;

   printf("Enter the number of elements you want to insert : ");
   scanf("%d", &n);

   for (i = 0; i < n; i++)
   {
      printf("Enter element %d : ", i + 1);
      scanf("%d", &Arr[i]);
      sum += Arr[i];
   }

   printf("\nThe sum of the array is : %d", sum);
   printf("\nThe average of the array is : %0.2f", (float)sum / n);

   return 0;
}
```

```
Enter the number of elements you want to insert : 6
Enter element 1 : 10
Enter element 2 : 20
Enter element 3 : 15
Enter element 4 : 18
Enter element 5 : 21
Enter element 6 : 19

The sum of the array is : 103
The average of the array is : 17.17
```

4. Write program to print Fibonacci series with and without function.

Fibonacci Series in C

Fibonacci Series in C: In case of fibonacci series, *next number is the sum of previous two numbers* for example 0, 1, 1, 2, 3, 5, 8, 13, 21 etc. The first two numbers of fibonacci series are 0 and 1.

There are two ways to write the fibonacci series program:

- Fibonacci Series without recursion
- Fibonacci Series using recursion

Fibonacci Series in C without recursion

```
#include<stdio.h>
int main()
{
 int n1=0,n2=1,n3,i,number;
 printf("Enter the number of elements:");
 scanf("%d",&number);
 printf("\n%d %d",n1,n2);//printing 0 and 1
 for(i=2;i<number;++i)//loop starts from 2 because 0 and 1 are already printed
 {
  n3=n1+n2;
  printf(" %d",n3);
  n1=n2;

  n2=n3;

 }

  return 0;

 }
```

Output:

```
Enter the number of elements:15

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Fibonacci Series using recursion in C

```
 #include<stdio.h>

void printFibonacci(int n){

   static int n1=0,n2=1,n3;

   if(n>0){

        n3 = n1 + n2;

        n1 = n2;

        n2 = n3;

        printf("%d ",n3);

        printFibonacci(n-1);

   }

}

int main(){

   int n;

   printf("Enter the number of elements: ");

   scanf("%d",&n);

   printf("Fibonacci Series: ");

   printf("%d %d ",0,1);
```

```
    printFibonacci(n-2);//n-2 because 2 numbers are already printed

  return 0;

 }
```

Output:

```
Enter the number of elements:15

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

6. What is recursion? Explain with an example.

Recursion is the process of repeating items in a self-similar way. In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function.

```
void recursion() {
  recursion(); /* function calls itself */
}

int main() {
  recursion();
}
```

The C programming language supports recursion, i.e., a function to call itself. But while using recursion, programmers need to be careful to define an exit condition from the function, otherwise it will go into an infinite loop.

Recursive functions are very useful to solve many mathematical problems, such as calculating the factorial of a number, generating Fibonacci series, etc

Example :

Number Factorial

The following example calculates the factorial of a given number using a recursive function −

```
#include <stdio.h>

unsigned long long int factorial(unsigned int i) {

  if(i <= 1) {
    return 1;
  }
  return i * factorial(i - 1);
}

int  main() {
   int i = 12;
   printf("Factorial of %d is %d\n", i, factorial(i));
   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Factorial of 12 is 479001600
```