Q1. Fill in the blanks. **[0.5x4=2]**

a) By default, return type of a function is **int**.

b) ***** is indirection operator.

c) Size of float & double is **4** & **8** bytes respectively.

d) Result of 5<<2 & 12>>2 is **20** & **3** respectively.

Q2. Write the output/error of the following code. **[0.5*6=3]**

a) `int x = 0;`
`if(x = 0 ) printf("True");`
`else printf("False");`
	b) `int a=2*3/4+2.0/5+8/5;`
	`printf("%d", a);`
		c) `int a[ ] = {10, 11, 12, 13, 14};`
		`printf("%d %d %d", *a, a[0], a[4]);`

d) `int a = 0, b = 1, c, d, e;`
`c = b++ && (d=b);`
`e = ++a || ++c;`
`printf("%d %d %d %d %d", a, b, c, d, e);`
	e) `int i = -6;`
	`while(i) {`
	`printf("%d", i++);`
	`}`
		f) `int f = 0; switch ( f ) {`
		`case 0 :`
		`case 1 : printf ("1#\n") ;`
		`default: printf ("2#") ; }`

Ans: a) False b) 2 c) 10 10 14 d) 1 2 1 2 1 f) -6-5-4-3-2-1 e) 1# 2#

Q3. a) Differentiate between variable and keywords as well as variable declaration and variable definition. Also write the rules for naming a variable in C.

int var = 10;

The statement written above is a variable declaration and definition. Var is defined to have 10. The keyword here is **int**. In case of variables, the keywords used signifies the type of data that it is going to hold, and the space required to store that data.

Rules for naming a variable:

1. Characters Allowed to used in name:

- Underscore(_)
- Capital Letters (A – Z)
- Small Letters (a – z)
- Digits (0 – 9)

2. First Character of the name should be **alphabet or Underscore**

3. Variable name should not be a **Reserved (Keyword) word**

Ex: **Valid Names: Num, Num1, _NUM, NUM_temp2**

Invalid Names: number 1, num 1, 1num, 1_num, 1, int

b) Explain function call by reference and call by value with appropriate examples.

Ans:

Call by Value:

In call by value, a copy of actual arguments is passed to formal arguments of the called function and any change made to the formal arguments in the called function have no effect on the values of actual arguments in the calling function.

In call by value, actual arguments will remain safe, they cannot be modified accidentally.

Example: Call by Value:

```
#include <stdio.h>
void swapByValue(int, int); /* Prototype */
int main() /* Main function */
{
 int n1 = 10, n2 = 20;
 /* actual arguments will be as it is */
 swapByValue(n1, n2);
 printf("n1: %d, n2: %d\n", n1, n2);
}
void swapByValue(int a, int b)
{
 int t;
 t = a; a = b; b = t;
}
OUTPUT:
======
n1: 10, n2: 20
```

The `swapByValue()` does not affect the arguments `n1` and `n2` in the calling function it only operates on `a` and `b` local to `swapByValue()` itself.

Call by Reference/Address:

In call by reference, the location (address) of actual arguments is passed to formal arguments of the called function. This means by accessing the addresses of actual arguments we can alter them within from the called function.

In call by reference, alteration to actual arguments is possible within from called function; therefore the code must handle arguments carefully else you get unexpected results.

Example: Reference/Address:

```
#include <stdio.h>
void swapByReference(int*, int*); /* Prototype */
int main() /* Main function */
{
 int n1 = 10, n2 = 20;
 /* actual arguments will be altered */
 swapByReference(&n1, &n2);
 printf("n1: %d, n2: %d\n", n1, n2);
}
void swapByReference(int *a, int *b)
{
 int t;
 t = *a; *a = *b; *b = t;
}
OUTPUT
======
n1: 20, n2: 10
```

In call by reference, to pass a variable `n` as a reference parameter, the programmer must pass a pointer to `n` instead of `n` itself. The formal parameter will be a pointer to the value of interest. The calling function will need to use `&` to compute the pointer of actual parameter. The called function will need to dereference the pointer with `*` where appropriate to access the value of interest.

c) A 2-D array **array[100][50]** contains floating values. The base address of the array is 800, what will be the address of the element present at **[60][40]**? [Size of float is 4 bytes]
Ans: 800 + (4 * 50 * 60) + (4 * 40) = 800 + 12000 + 160 = 12960

[2+2+1=5]

Q4. Answer the following. **[2+2+1=5]**

a)Explain bitwise operators with example?
Ans: (Any 4 will also be enough)

1. & (bitwise AND) Takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.

```
int a = 12, b = 25;
int c = a & b;
```

Explanation:

12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bit Operation of 12 and 25
 00001100
& 00011001

 ____
 00001000 = 8 (In decimal)

So, C will contain 8

2. | (bitwise OR) Takes two numbers as operands and does OR on every bit of two numbers. The result of OR is 1 any of the two bits is 1.

```
int a = 12, b = 25;
int c = a & b;
```

Explanation:

12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bit Operation of 12 and 25
 00001100
| 00011001

 ____
 00011101 = 29 (In decimal)

So, C will contain 29

3. ^ (bitwise XOR) Takes two numbers as operands and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.

```
int a = 12, b = 25;
int c = a & b;
```

Explanation:

12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bit Operation of 12 and 25
 00001100

^ 00011001

00010101 = 21 (In decimal)

So, C will contain 21

4. **<<** (left shift) Takes two numbers, left shifts the bits of the first operand, the second operand decides the number of places to shift.

```
int a = 12;
int c = a << 2;
```

Explanation:

12 = 00001100 (In Binary)

Bit Operation of 12 << 2

00001100**00** = 48 (In decimal) Here two additional bits are placed at the beginning i.e. LSB positions.

So, C will contain 48

5. **>>** (right shift) Takes two numbers, right shifts the bits of the first operand, the second operand decides the number of places to shift.

```
int a = 12;
int c = a >> 2;
```

Explanation:

12 = 00001100 (In Binary)

Bit Operation of 12 >> 2

00001~~00~~ = 000011 = 3 (In decimal) Here two bits are removed from beginning i.e. LSB positions.

So, C will contain 3

6. ~ (bitwise NOT) Takes one number and inverts all bits of it

```
int a = 12;
int c = ~a;
```

Explanation:

12 = 00001100 (In Binary)

Bit Operation of ~12

11110011 = Going to be a large decimal number, since all the bits will be inverted (i.e. all 32 bits since it is an integer)

b) A program is written below to find whether it is a given number is prime or not. Fill the appropriate value/text at all empty places marks.

```
#include <stdio.h>
void main( ) {
   int n, i, flag = 0;
   printf("Enter a positive integer: ");
   scanf("%d", _____&n_____ );
   for(i = ____2____ ; i <= n/2; i = ___i++_____ ) {
      if( n ___%___ i == 0)        {        flag = ___1___;          break;       }
   }
   if ( __flag = = 1 _____ )          printf("Not Prime");
   else          printf("Prime");
}
```

c) Is C a case sensitive language? Justify your answer.

Ans: In C programming language both capital and small letters are considered different.

For example: total and Total will be two different variables and hence cannot be used interchangeably. Similarly, all C keywords can only be written in small letters since it has been declared/defined in small letters only.

Roll No: [ ]

Q5. a) Differentiate between these with respective flowcharts.

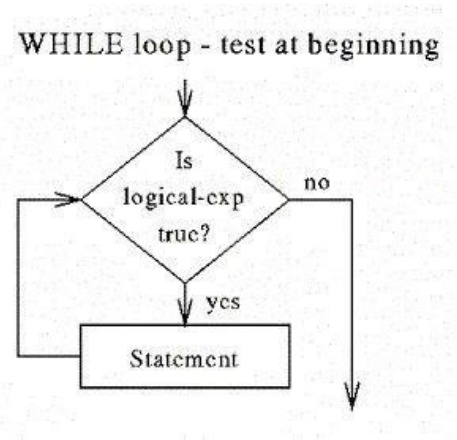
i.) while and do-while

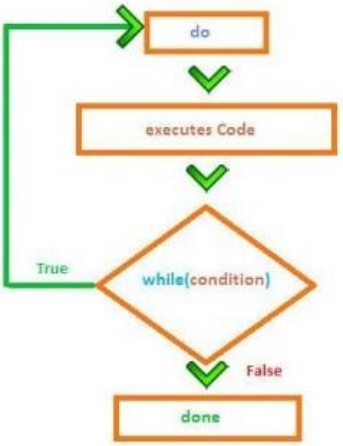
ii.) if-else and switch

Ans: a) Difference between while loop & do-while loop

	While	Do While
1.	This is an entry-controlled loop.	This is an exit-controlled loop.
2.	The body of the loop will execute Zero or more number of times.	The body of the loop will execute at least once.

3.





b) Difference between if-else & switch-case

1. In *else if ladder*, the control goes through the every else if statement until it finds true value of the statement or it comes to the end of the else if ladder. In case of *switch case*, as per the value of the switch, the control jumps to the corresponding case.

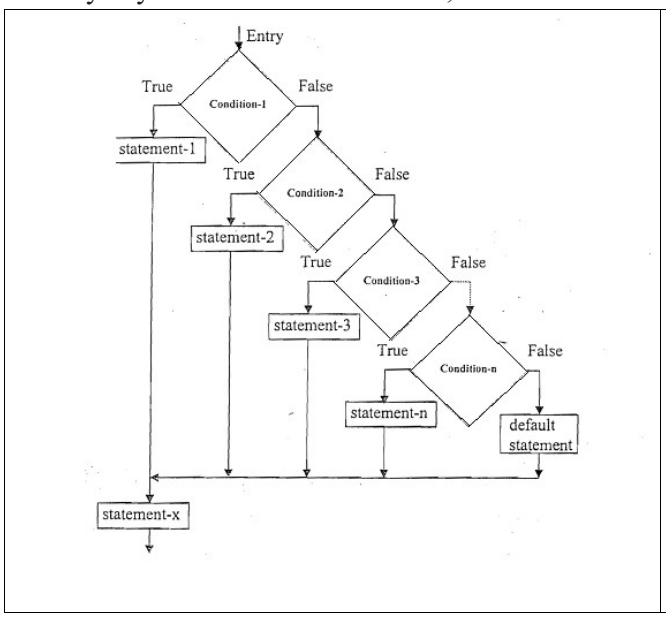
2. The switch case is more compact than lot of nested else if. So, switch is considered to be more readable.

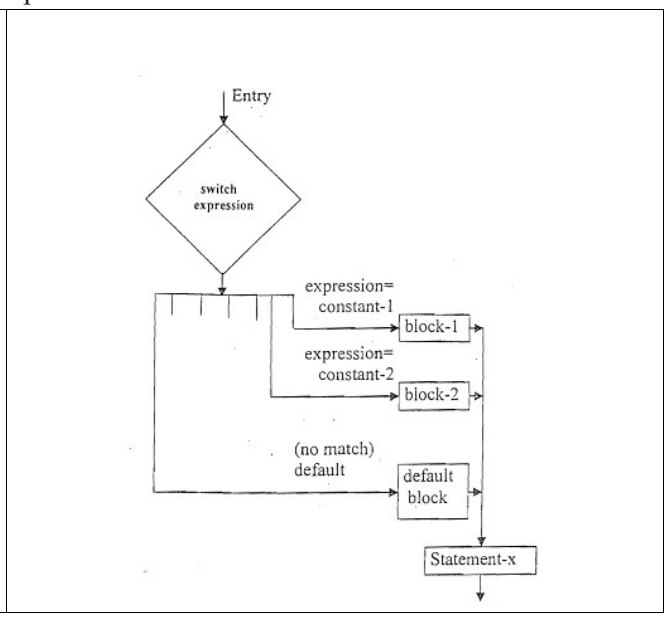
3. The use of break statement in switch is essential but there is no need of use of break in else if ladder.

4. The variable data type that can be used in expression of switch is integer as well as character(variable or Constants) only where as in else if ladder accepts other types also.

5. *Switch case statement* work on the basis of equality operator whereas *else if ladder* works on the basis of true false(zero/non-zero) basis.

6. Each case in switch statement is independent of the previous one I.e two cases can not be same in anyway. In case of else if ladder, there can be multiple similar condition test.





b) Write a program to swap the content of two integer variables without using temporary/third variable.

Ans: Any one of these three or some other similar logic [Explanation is not required].

```
#include <stdio.h>
void main()
{
  int x = 10, y = 5;
  // Code to swap 'x' and 'y'
  x = x + y;  // x now becomes 15
  y = x - y;  // y becomes 10
  x = x - y;  // x becomes 5
  printf("After Swapping: x = %d, y = %d", x, y);
}
Output: After Swapping: x = 5, y = 10
```

```
#include <stdio.h>
void main()
{
  int x = 10, y = 5;
  // Code to swap 'x' and 'y'
  x = x * y;  // x now becomes 50
  y = x / y;  // y becomes 10
  x = x / y;  // x becomes 5
  printf("After Swapping: x = %d, y = %d", x, y);
}
Output: After Swapping: x = 5, y = 10
```

```
#include <stdio.h>
void main()
{
    int x = 10, y = 5;
    // Code to swap 'x' (1010) and 'y' (0101)
    x = x ^ y; // x now becomes 15 (1111)
    y = x ^ y; // y becomes 10 (1010)
    x = x ^ y; // x becomes 5 (0101)
    printf("After Swapping: x = %d, y = %d", x, y);
}
Output: After Swapping: x = 5, y = 10
```

[4+1=5]

Q6. a) Write a program to input two integer arrays of size 5 each. Sum the elements index wise and print the result. For example, if array **A** contains 1, 0, 4, 3, 5 and array **B** contains 6, 1, 2, 8, 3 then the output should be 7, 1, 6, 11, 8.

```
#include<stdio.h>
void main( ){
int A[5], B[5], i;
for (i=0;i<5;i++)
    scanf("%d",&A[i]);
for (i=0;i<5;i++)
    scanf("%d",&B[i]);
```

```
for (i=0;i<5;i++)
    printf(" %d ",A[i]+B[i]);
}
```

b) What are user-defined and library functions, explain with example? Is main a library function?

Ans:

Library functions are those functions which are already defined in C library, example `printf()`, `scanf()`, `strcat()` etc. You just need to include appropriate header files to use these functions. These are already declared and defined in C libraries. These functions are also called as built-in functions.

User-defined functions on the other hand, are those functions which are defined by the user at the time of writing program. These functions are made for code reusability and for saving time & space.

```
#include <stdio.h>
void swapByReference(int*, int*); /* Prototype */
int main() /* Main function */
{
 int n1 = 10, n2 = 20;
 /* actual arguments will be altered */
 swapByReference(&n1, &n2);
 printf("n1: %d, n2: %d\n", n1, n2);
}
void swapByReference(int *a, int *b)
{
 int t;
 t = *a; *a = *b; *b = t;
}
OUTPUT
======
n1: 20, n2: 10
```

Here printf is a library/built-in function which is declared in stdio.h library/header file and swapByReference is user defined function. A point to highlight is that, in library/built-in functions we do not need to define it, just name of the header file where it is declared it and that call to it with proper number/type of argument(s). But this is not the case with user-defined functions, we need to explicitly write it.

Main is not a library function. Since it is defined by user, it is a user defined function.

c) How to declare a generic pointer?

Ans:

void *ptr;

Here ptr is a pointer to void i.e. it can refer to any type of data. So it is called a generic pointer.

[2+2+1=5]