Qu. How to reuse class. Explain with example.

Classes are used to maintain the code and reduce the duplication of same code again. You can define your own class for a specific task. The best example of reusing class is OOPS (Object-Oriented Programming).

Class contain several parts ⇒ i) struct ii) initialize function iii) main function.

Eg ⇒

```
# include <stdio.h>
typedef struct {
    int num1,
    int num2,
} Number Adder;

void initialize (Number Adder * adder, int n1, int n2) {
adder → num1 = n1
adder → num2 = n2
}

    int addNumber (NumberAdder * adder) {
    return adder → num1 + adder → num2,
    }
int main() {
NumberAdder ins1;
initialize (& ins1, 10, 20);
int result = addNumbers (ins2);

printf ("Result : %d\n", result);
```

Que38. Define tree. List the tree traversal techniques

⇒ Tree is like a hierarchical data structure containing roots and several sub-branch or children. These children can also have their sub-branch forming a tree like structure.

There are techniques called traversal techniques to visit the nodes of the tree. Some of the examples are-

1. Inorder Traversal
2. Preorder Traversal
3. Postorder Traversal

Que 39 What is an adjacency list. When it is used?

An adjacency list is list of vertices points of a graph. An adjacency list is used to parse a graph from the vertices stored in the list. In simple words, the adjacency list is used to store and manage a graph's edges vertices and later a graph can be generated with the help of adjacency list.

Some function to perform the task-

1. newAdjListNode(): creat a new adjacency node.
2. createGraph(): Initialize a graph with give no. of vertices.
3. addEdge(): Add edge to a graph.
4. printGraph(): Prints the graphs.

Q40

Write a program to convert infix to postfix using stack.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

struct Stack {
    int top;
    unsigned capacity;
    char* array;
};

struct Stack* createStack(unsigned capacity) {
struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
stack->capacity = capacity;
stack->top = -1
stack->array = (char*)malloc(stack->capacity * sizeof(char));
    return stack;
}

void infixToPostfix(char* infix) {
    struct Stack* stack = createStack(strlen(infix));
    int i, k;
    for(i = 0, k = -1; infix[i]; ++i) {
        if(isOperand(infix[i]))
        infix[++k] = infix[i];

else if(infix[i] == '(')
    push(stack, infix[i]);
```

Q40 Part-2

```
else if ( infix[i] == ')'){
  while (!isEmpty(stack) && peep(stack) != '(')
    infix[++R] = pop(stack);
      pop(stack);
}
  else {
while(!isEmpty(stack) && precedence(infix[i]) <=
    precedence(peek(stack)))

  infix[++R] = pop(stack);
    push(stack, infix[i]);
}
}

  while(!isEmpty(stack))
      infix[++R] = pop(stack);
infix[++R] = '\0';
    printf("Postfix %s\n", infix);
}
```

Que 41 Explain in detail the routine for Depth first and breadth first traversal.

The Depth first helps to analyze a graph. It check each and every node or branch of the graph. It is a fundamental algorithm to search and analyze a graph.
It's start searching from the root of the structure and searches each and every branch.
Whereas the breadth first traversal is somewhat different. It also starts from the bottom or root of the node but instead of going to other branches it checks all the current branches of current level.

Q40

Write a program to convert infix to postfix using stack.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

struct Stack {
    int top;
    unsigned capacity;
    char* array;
};

struct Stack* createStack(unsigned capacity) {
struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
stack->capacity = capacity;
stack->top = -1
stack->array = (char*)malloc(stack->capacity * sizeof(char));
    return stack;
}

void infixToPostfix(char* infix) {
    struct Stack* stack = createStack(strlen(infix));
    int i, k;
    for (i = 0, k = -1; infix[i]; ++i) {
        if (isOperand(infix[i]))
            infix[++k] = infix[i];

else if (infix[i] == '(')
    push(stack, infix[i]);
```