Hello , **KloverCloud**

My self, Md. Mahmudul Hasan Shahin

Here, Answer of all written Question are given below .

Answer of Questions

Question_1:

Ans :

- By concept of OOP class of **Circle** can be create as (Please check **Question_1.cpp**):

```
class Circle : public Area
{
public:
        float rad;
        Circle(float r)                         // Constructor
        {
                rad = r;
        }
        void area_calc()
        {
                cout<<"Area of Circle    : "<<3.1416 * (rad*rad)<<endl;
        }
};
```

- And **Rectangle** class look like :

```
class Rectangle : public Area
{
public:
        float a, b;
        Rectangle(float x, float y)                    // Constructor
        {
                a = x;
                b = y;
        }
        void area_calc()
        {
                cout<<"Area of Rectangle : "<<a * b<<endl;
         }
};
```

- Runtime Polymorphism can be achieved by Method overriding as below (Please check Question_1.cpp).

```
class Area
{
public:
        void area_calc()
        {
                cout << "Base Class Area :) \n" ;
        }
};
class Circle : public Area
{
public:
        void area_calc()
        {
                cout<<"Area of Circle    : "<<3.1416 * (rad*rad)<<endl;
        }
};
```

Question_2:

Ans :

<u>Stack:</u>

Stack follows LIFO (Last In First Out) style for data push and pop. Stack can be created temporally by function then it is a temporary storage which is cleared/erased after end of execution.

- Data access speed is high.
- Only used as local variable.
- Variable size can't be resized.
- Memory limit is short.
- Faster and Linear data structure.

<u>Heap:</u>

Heap follows hierarchical data structure which can allocate memory dynamically . All global variable are stored in heap.

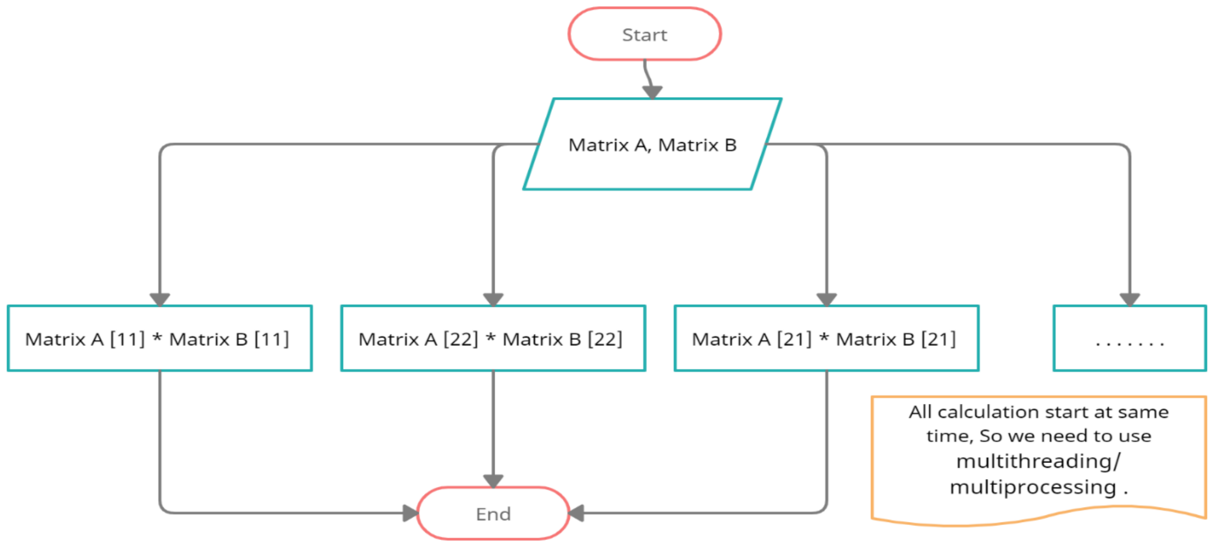
- Slower Data access speed.
- Automatically used for global variable allocation.
- Variable size can be resized.
- Memory can be fragmented.
- Slower and Hierarchical data structure.

According to memory size we should use **Heap**. On the other hand if we consider speed then Stack should use.

Question_3:

Ans :

If we need to improve or speedup calculation speed, we can implement multithreading/multiprocessing.


Start
Matrix A, Matrix B
Matrix A [11] * Matrix B [11]
Matrix A [22] * Matrix B [22]
Matrix A [21] * Matrix B [21]
. . . .
All calculation start at same time, So we need to use multithreading/ multiprocessing .
End

In multiprocessing we know all instruction execute parallelly. So all matrix calculation can be execute at a time.

Please check **Question_3.py** [I never implement thread in C++ so here I use Python] for better understanding.

Question_3:

Ans :

- If we need to use global count variable then the correct recursive function look like:

```
void traverse(struct Node* node)
{
        if (node == NULL){return;}
        count ++;
        traverse(node->left);
        traverse(node->right);
}
```

- Otherwise we can calculate total number of nodes using local count variable as below :

```
int traverse(struct Node* node)
{
        int count = 0;
        if (node == NULL){return 0;}
        count ++;
        count += traverse_2(node->left);
        count += traverse_2(node->right);
        return count;
}
```

Please check **Question_4.cpp** for better understanding.