Answer Script

Question No. 1-a

Stack Memory: Stack/Static memory is called a compile time memory. When we normally create a variable it saves on stack memory. Like int a = 10. Stack memory works sequentially. It follows the activation record, here the last thing will work first. Stack memory has some limitations, first problem is if we create some function and call one from another it will be called sequentially and after the work function is removed automatically.Then we can’t access the value of the function or change the value of that function any more and second problem is we can’t change the memory or size after initialising it one time because it’s size is fixed.

Heap Memory: Heap/Dynamic memory is called a runtime memory. Stack memory has some limitations and that’s why heap memory arrives with the solution.It works as rosource. Heap memory gives us the permission to change the memory access even after initialization. For example we can increase the size of an array by heap memory. Here we can delete a memory to save the unnecessary memory and one of the most benefit of the heap memory is it saves the data even after the function runs out. We can access the memory address even after the function call and return. We can declare a data to heap memory like this, int *a = new int a[5]; here we have used the new keyword to save data on heap memory and it is giving the address of the data to a pointer variable which is created in a stack memory. By initialising data to heap memory we can return the address of the array and use it even after the function call runs out by dereferencing the pointer variable. Therefore, heap memory is useful in many ways

Answer No. 1-b

We need dynamic memory allocation for many reasons. One of them is to use the value of the functions after it returns. If we use static memory we can not reinitialize the size of a variable. Stack memory gives us a fixed memory but dynamic memory gives us the access to change or increase the memory any time.The data of the dynamic memory will not automatically delete after function call and we can also delete unnecessary things after our work.

We need dynamic memory allocation to use the value even after the work of a function is done. For example:

```cpp
#include <bits/stdc++.h>
using namespace std;

int* fun(int n)
{
    int *a = new int[n];

    for (int i = 0; i < n; i++)
    {
        cin >> a[i];
    }

    return a;
}

int main()
{
    int n=5;

    int *a = fun(n);

    for (int i = 0; i < n; i++)
    {
        cout << a[i] << " ";
    }

    return 0;
}
```

If we don’t use heap memory we will get the address of the array but we can’t access it because stack memory releases everything after finishing the work of it. So to return an array heap memory must be used.

Also for increasing the size we use heap memory because it can delete unnecessary things from it by our command. For example:

```cpp
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int* a = new int[5];
    for (int i = 0; i < 5; i++)
    {
        cin >> a[i];
    }

    int *b = new int[7];

    for (int i = 0; i < 5; i++)
    {
        b[i] = a[i];
    }

    b[5] = 30;
    b[6] = 35;

    for (int i = 0; i < 7; i++)
    {
        cout << b[i] << " ";
    }

    //delete a array;
    delete[] a;  // now deleting the unnecessary array a[5]
    return 0;
}
```

By this we can save memory and change the size of an array. So dynamic memory is so important for some works.

Question No. 1-c

We can create a dynamic array by using **new** keywords. For example,

```
    int* a = new int[5];
    for (int i = 0; i < 5; i++)
    {
        cin >> a[i];
    }
```

Here a dynamic array has been created in heap memory and it is passing it's address to a, we can use the array as the same as an array of stack memory.

There are many benefits of creating a dynamic array. If an array is created dynamically we can create it to any of the functions and pass the address of it to the main function. We can access the address easily and get the values of the array which is not possible in static memory allocation.
Here is the example:

```
#include <bits/stdc++.h>
using namespace std;

int* fun(int n)
{
    int *a = new int[n];

    for (int i = 0; i < n; i++)
    {
        cin >> a[i];
    }

    return a;
}

int main()
{
    int n=5;
    int *a = fun(n);

    for (int i = 0; i < n; i++)
    {
        cout << a[i] << " ";
    }

    return 0;
}
```

Also we can increase the size of an array and delete the array when an array is created dynamically. It saves our memory and makes our work much easier. Here is the example:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int* a = new int[5];
    for (int i = 0; i < 5; i++)
    {
        cin >> a[i];
    }

    int *b = new int[7];

    for (int i = 0; i < 5; i++)
    {
        b[i] = a[i];
    }

    b[5] = 30;
    b[6] = 35;

    for (int i = 0; i < 7; i++)
    {
        cout << b[i] << " ";
    }

    //delete a array;
    delete[] a;  // now deleting the unnecessary array a[5]
    return 0;
}
```

So, there are many benefits of creating a dynamic array.

Question No. 2-a

A class and object is a concept of object oriented programming. A class works like a blueprint and an object works like the model of that blueprint. Class works as a user defined data type. We can create data types here as we want. There can be a mix of different data types in a class. We can create as many objects as we want which can save different data of the types which are initialised in a class. Declaring of a class and object is given below:

```
#include <bits/stdc++.h>
using namespace std;

class Student here we are creating user defined data type
          // which is a class, Student is the class name.
```

```
{
    public:
        char name[101];
        int roll;
        int cls;
        char section;
};

int main()
{
    Student asif; /// Student is a user defined data type
       Which is class name and asif is a variable of
     the data type which is an object///

    asif.roll = 18;
    asif.cls = 8;
    asif.section = 'B';
    char nm1[101] = "Asif Abdullah";
    strcpy(asif.name,nm1);

    //Printing:

    cout << "Asif's Information:" << endl;
    cout << asif.name << endl;
    cout << asif.roll << endl;
    cout << asif.section << endl;
    cout << asif.cls << endl<< endl;

    return 0;
}
```

Here Student class has an object which is named as asif.

Answer No. 2-b

A constructor is a special type of function of a class which is automatically called when an object is created of that class. We need a constructor to simplify the initialization of the values of an object. Constructor always have the same name as the class name and it does not have any return type and it is called automatically when an object is created.
We can create an constructor in this way:

```
#include <bits/stdc++.h>
using namespace std;

class Student
{
public:
    char name[101];
    int roll;
    int cls;
    char section;
    Student(char *nm, int r, int c, char s) // constructor
    {
        strcpy(name, nm);
        roll = r;
        cls = c;
        section = s;
    }
};

int main()
{
    Char nm[101] = "Asif Abdullah"
    Student asif(nm, 18, 8, 'B'); //passing the value to a
    // constructor with object creation.
    return 0;
}
```

Here we have created a constructor named Student which is in student class and passes the values of the object and initialises these values to the object very easily.

Question No. 2-c

```cpp
#include <bits/stdc++.h>
using namespace std;
class Person
{
    public:
    char name[101];
    float height;
    int age;

    Person(char *n,float h,int a)
    {
        strcpy(name,n);
        height = h;
        age = a;
    }
};

int main()
{
    char nm[101] = "Asif Abdullah";
    Person* asif = new Person(nm,5.4,21);

    cout << "Asif's Information:\n";
    cout << asif->name << endl;
    cout << asif->height << endl;
    cout << asif->age << endl;
    return 0;
}
```

Answer No. 3-a

The size that an object allocates to the memory depends on the data initialised in it.
The total size of all the data inside the object is the size of an object but the size of an empty object is 1 byte.

For example: Suppose we are making a vehicle class which has a char, and int and one double type of data, so 1 byte of char, 4 byte of int and 8 byte of double type data will be added to the total size of an object.

```
class vehicle{
public:
     char helper;
     int wheels;
    double number;
};
```

So 1+4+8 bytes will be added to the total size of an object.

Question No. 3-b

Yes, We can return a static object from a function.
Example is given below:

```
#include<bits/stdc++.h>
using namespace std;

class Student
{
    public:
        char name[100];
        int cls;
        int roll;
        char sec;
    Student(char *n,int c,int r,char s)
    {
        strcpy(name,n);
        cls = c;
        roll = r;
        sec = s;
    }



};

Student fun()
{
    char name[100] = "Asif Abdullah";
    Student asif(name,8,18,'B');

    return asif;
}

int main()
{
    Student asif2 = fun();
    cout << "Asif's Information:" << endl;
    cout << asif2.name << endl;
    cout << asif2.roll << endl;
    cout << asif2.sec << endl;
    cout << asif2.cls << endl << endl;

    return 0;
}
```

Here a static object is created in a fun function named asif which returns the asif object to the main function and it is received by class asif2 . Here fun function deletes everything after return but before that it is returning all the things of asif to asif2 object. Then the values are passed to asif2. The process of this passing is called RVO(Return value optimization). Then asif2 receives all the data of asif and we can access the values of asif2 objects which were passed by object asif.

Answer No. 3-c

We use -> (arrow sign) to access the values of dynamic object. We can only use this arrow sign when there is a group or object and there is a pointer variable which is needed to to be dereferenced.

For example:

```
#include <bits/stdc++.h>
using namespace std;

class Student
{
public:
    char name[101];
    int roll;
    int cls;
    char section;
    Student(char *n,int r,int c,int s)
    {
        strcpy(name,n);
        roll = r;
        cls = c;
        section = s;
    }
};

int main()
{
    char nm[101] = "Asif Abdullah";
    Student* asif = new Student(nm,18,8,'B'); //dynamic
object creation
```

```
    (*asif).roll = 10; // jodi kono kisu update korte chai
taholeo direfference korte hoy (dynamic object er khetre)
///

     asif->roll = 9;

    //arrow sign: (*asif).name = asif->name

    cout << asif->name << endl; //  jodi dynamic object
hoy(orthat pointer thake jake direfference korte hoy+grp ba
object hoy) taholei evabe arrow diye access kra jbe//
    cout << asif->roll << endl;


    return 0;
}
```

Here we are accessing the values of asif class with -> arrow sign. asif->roll is similar to (*asif).roll

Question No. 3-d

```
#include <bits/stdc++.h>
using namespace std;

class Person
{
    public:
    char name[101];
    float height;
    int age;

    Person(char *n,float h,int a)
    {
        strcpy(name,n);
        height = h;
        age = a;
    }
};

int main()
{
    char nm1[101] = "Asif Abdullah";
    Person asif(nm1,5.4,21);

    char nm2[101] = "Anisul Islam";
    Person masud(nm2,5.2,19);

    if(asif.age > masud.age)
    {
        cout << asif.name << endl;
    }
    else if(asif.age < masud.age)
    {
        cout << masud.name << endl;
    }
    else{
        cout << "Both has same age\n";
    }

    return 0;
}



/*With dynamic object:
```

```cpp
#include <bits/stdc++.h>
using namespace std;

class Person
{
    public:
    char name[101];
    float height;
    int age;

    Person(char *n,float h,int a)
    {
        strcpy(name,n);
        height = h;
        age = a;
    }
};

int main()
{
    char nm1[101] = "Asif Abdullah";
    Person* asif = new Person(nm1,5.4,21);


    char nm2[101] = "Masud Abdullah";
    Person* masud = new Person(nm2,5.2,19);

    if(asif->age > masud->age)
    {
        cout << asif->name << endl;
    }
    else if(asif->age < masud->age)
    {
        cout << masud->name << endl;
    }
    else{
        cout << "Both has same age\n";
    }

    return 0;
}

*/
```