Answer Script

Question No. 1-a

Explain Stack and Heap memory.

Answer No. 1-a

In general, 2 types of memory -

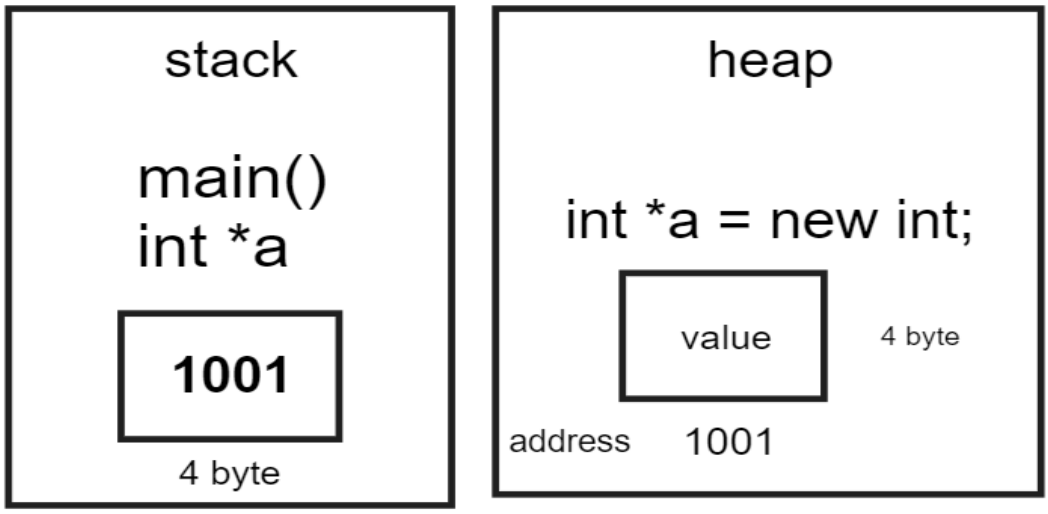
1. Heap Memory
2. Stack Memory

Stack Memory(static Memory):

The stack is optimized for fast access and efficient memory.
Stack is used to store data, local variables, function parameters, return addresses, and function calls.
Also called compile-time memory.
The stack is a last-in-first-out (LIFO) data structure(Linear).
Memory is allocated in a contiguous block(Sequentially).
The allocation and deallocation of stack memory are handled automatically by the compiler.

Heap Memory(dynamic Memory):

Heap memory is used for dynamic allocation of data, and store objects. It can't be accessed directly & slower memory.
The allocation and deallocation of heap memory are controlled by the programmer. Also called run-time memory.
Memory is allocated in any random order(hierarchical).
The heap is a first-in-first-out (FIFO) data structure.
Heap can grow and shrink dynamically during program execution as memory.
Heap memory can lead to memory leaks, when allocated memory is not deallocated properly.



Question No. 1-b

Why do we need dynamic memory allocation? Explain with examples.

Answer No. 1-b

Explanation of Dynamic Memory Allocation:

The process of allocating memory at runtime is called dynamic memory allocation.
When declaring a very large array, which most of them don't need. Sometimes we declare a very small array, but after running the program may need a larger size. And to solve these problems is dynamic memory allocation.
Dynamic memory allocation allows memory from the heap (new keyword).
Dynamic memory allocation is more efficient as compared to Static memory allocation.
In this process, while executing a program, the memory can be changed.
Allocated memory can be released at any time during the program (delete keyword).

Example:

** Array allocated in heap memory and return array.
** Save the return value and print it using the index of the array.

```
//code
#include <bits/stdc++.h>
using namespace std;
int *fun(int n)
{
    // create dynamic array
    int *a = new int[n];
    for (int i = 0; i < n; i++)
    {
        // input value
        cin >> a[i];
    }
    // return array
    return a;
}
int main()
{
    int n;
    cin >> n;
    // receive array as a pointer
    int *ar = fun(n);
    for (int i = 0; i < n; i++)
    {
        // print value
        cout << ar[i] << " ";
    }
    return 0;
}
```

Question No. 1-c

How to create a dynamic array? What are the benefits of it?

Answer No. 1-c

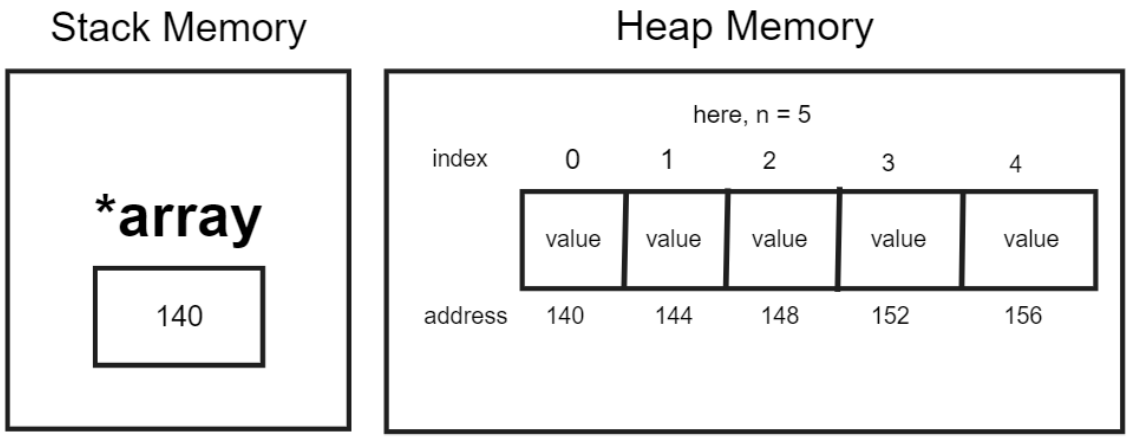
Create Dynamic Array:

In C++, create a dynamic array using the new keyword. Here's an example of how to do it:

//syntax create dynamic array

int* array = new int [n];

In this case, n represents the size of a dynamic array. The new keyword dynamically allocates in heap memory for an array of integers of size n and returns a pointer to the first element of the array.



Benefits of Dynamic Array:

Dynamic array can store elements of any data type.
Dynamic array support efficient insertion and deletion operations.
Dynamic array provides random access to elements.
Dynamic array allocates memory at runtime.
Dynamic array size can be changed during program execution.
Dynamic arrays are relatively easy to use.
Heap memory has no limitation for the allocated array element.
when an array element is stored in heap memory, the element remains in memory until deleted.

Example:

** Array a declare dynamically then insert value.
** Increase size create array b which copies all elements of an array.
** Then delete array a which are all value erase from heap memory. Print b array element.
** Also save memory when delete array a.

```
//code
#include <bits/stdc++.h>
using namespace std;
int main()
{
    // create dynamic array
    int *a = new int[5]{10, 20, 30, 40, 50};
```

```
    // dynamically increase array size
    int *b = new int[7];
    // copy element from a to b
    for (int i = 0; i < 5; i++)
    {
        b[i] = a[i];
    }
    b[5] = 60, b[6] = 70;
    // delete array a
    delete[] a;
    for (int i = 0; i < 7; i++)
    {
        cout << b[i] << " ";
    }
    return 0;
}
```

Question No. 2-a

How does class and object work? How to declare an object?

Answer No. 2-a

Work method of class and object:

User defined data type.
Object-oriented programming (OOP) includes class and object.
Class is a blueprint or template for creating objects, while an object is an instance of a class.
The object is a model of a class.
A class could be used as a template of a car.
Template store properties, such as model, wheel number, color, Numbers of seat.
object create follows the properties of a class.
class is a group of data, which is declared as a data type.
Objects act like variables of the data type.
Group data type(class) have access modifiers such as public, private, protected.

Example:

** Example of Class
// create a class
// class name is Car

```
//code
class Car
{
public:
    char model[20];
    char color[10];
    int wheel;
    int seat;
};
```

** Example of Object

```
//code
int main()
{
    // Car is a data type
    //  object is a variable
    //  variable name is audi
    Car audi;
    // create audi model car
    audi.wheel = 4;
    audi.seat = 5;
    char m[20] = " Audi A8";
    strcpy(audi.model, m);
    char c[10] = "gray";
    strcpy(audi.color, c);
    // print model
    cout << audi.seat << endl;
    cout << audi.color << endl;
}
```

Question No. 2-b

What is a constructor and why do we need this? How to create a constructor show with an example.

Answer No. 2-b

Constructor:

constructor is a special type of function.
A constructor is a special method within a class.
Constructor have same name as the class and both are include class scope.
Constructor has no return type.
Two types of constructors: default constructors. parameterized constructors.

Default constructors have no parameters, and initialize the object data members to default values.
Parameterized constructors have parameters, and allow initialize the object data members to specific values.

constructor need because -

It is called automatically when the object is instantiated.
Its purpose is to set the initial state of the object by assigning values.
Then the value passes the object into the constructor.
constructor receives value as parameter which assigns the parameter in a class variable.
In class scope class variable and constructor parameter name can't be same.
It can be overloaded, which means that it defines multiple constructors with different parameter lists.

Example:

** create constructor in class scope

```
//code
class Cricketer
{
public:
    int jersey_no;
    char country[100];

     // create constructor
    // j, *c as parameter

    Cricketer(int j, char *c)
    {
        // parameter value assign in class variable

        jersey_no = j;
        strcpy(country, c);
    }
};
```

** create object for passing value to constructor

```
//code
int main()
{
    char k[100] = "India";
    // passing value for constructor
    Cricketer kohli(29, k);
    // output
    cout << kohli.country << endl;
    return 0;
}
```

Question No. 2-c

Create a class named **Person** where the class will have properties name(string), height(float) and age(int). Make a constructor and create a dynamic object of that class and finally pass proper values using the constructor.

Answer No. 2-c

** Class name **Person**, Constructor **Person**, Dynamic object **Khaled**.

```
//code
#include <bits/stdc++.h>
using namespace std;
class Person
{
public:
    // class properties
    char name[20];
    float height;
    int age;
    // constactor
    Person(char *n, float h, int a)
    {
        // copy string n to name
        strcpy(name, n);
        height = h;
        age = a;
    }
};
int main()
{
    char nm[20] = "Khaled Hasan";
    // create dynamic object
    Person *khaled = new Person(nm, 5.7, 22);
    // check output
    cout << khaled->name << endl;
    cout << khaled->height << endl;
    cout << khaled->age << endl;
    return 0;
}
```

Question No. 3-a

What is the size that an object allocates to the memory?

Answer No. 3-a

Size of object allocate to memory:

# The size of an object in memory depends on the programming language, the data types, and the sizes of its attributes.
# Objects that contain data types such as integer, char, float, and double.
# which are smaller than objects that contain reference types, such as strings and objects.
# The size of an object is determined by the sum of the sizes of its attributes.
# Each attribute's size depends on its data type.
# For example,
integer = 4 byte
boolean = 1 byte
an object with an integer attribute and a boolean attribute would occupy 5 bytes (4 bytes + 1 byte).

Empty object is an object that does not have any data members. Empty objects can be used to hold a reference to another object. Empty object size of at least 1 byte which is greater than zero(>0).

Question No. 3-b

Can you return a static object from a function? If yes, show with an example.

Answer No. 3-b

Yes, return a static object from a function which following RVO(return value optimization) system. Static objects act as normal variables which are the return value.
But the function erases in stack memory when return an object. Main function receive the return object when calling the static object function.

Example:

```
//code
#include <bits/stdc++.h>
using namespace std;
class Student
{
public:
    char name[20];
    int roll;
    int cls;
    char section;
    Student(char *n, int r, int c, char s)
    {
        strcpy(name, n);
        roll = r;
        cls = c;
        section = s;
    }
};
// create return type function
// return type Student
Student fun()
{
    char nm[20] = "Rakib Hasan";
    // create static object
    Student rakib(nm, 29, 9, 'A');
    // return object
    return rakib;
}
int main()
{
    // static object receive return value
    Student rakib = fun();
    // check output
    cout << rakib.name << endl;
    cout << rakib.roll << endl;
    cout << rakib.cls << endl;
    cout << rakib.section << endl;
    return 0;
}
```

Question No. 3-c

Why do we need -> (arrow sign)?

Answer No. 3-c

Create a dynamic object when memory is allocated for the object in heap.
Heap return a address, which receive pointer object(*dhoni).
Dhoni is a pointer object.
Dynamic object contains the address where stored the data.
Then data gets from the heap when applying the dereferencing technique.
Two way access the data use dynamic object:
1. (*dhoni).country;
2. dhoni -> country;
Shortcut way get value: dhoni -> country;

Example:

```
//code
#include <bits/stdc++.h>
using namespace std;
class Cricketer
{
public:
    int jersey_no;
    char country[100];
};
int main()
{
    Cricketer *dhoni = new Cricketer;
    char c[100] = "India";
    strcpy(dhoni->country, c);
    dhoni->jersey_no = 29;
    cout << dhoni->jersey_no << endl;
    cout << dhoni->country << endl;
    return 0;
}
```

Question No. 3-d

Create two objects of the **Person** class from question **2-c** and initialize them with proper value. Now compare whose age is greater, and print his/her name.

Answer No. 3-d

** Two dynamic object ***hasan, *abdul**

```
//code
#include <bits/stdc++.h>
using namespace std;
class Person
{
public:
    // class properties
    char name[20];
    float height;
    int age;
    // constactor
    Person(char *n, float h, int a)
    {
        // copy string n to name
        strcpy(name, n);
        height = h;
        age = a;
    }
};
int main()
{
    char nm[20] = "Hasan Mia";
    char rn[20] = "Abdul Mia";
    // create dynamic object
    Person *hasan = new Person(nm, 5.7, 32);
    Person *abdul = new Person(rn, 5.9, 30);
    // check which one large
    (hasan->age > abdul->age) ? cout << hasan->name : cout << abdul->name;
    return 0;
}
```