**Answer Script**

| Question No. 1-a |
| --- |
| Explain Stack and Heap memory. |
| Answer No. 1-a |
| ***Stack Memory:*** In computer programming memory is used to store values or data of a program or function for a significant amount of time. Stack memory is a compile time memory that allocates the memory at runtime. Stack memory holds a function's values, variables and data till the function returns.  The functions all values, variables and data gets erased from the memory when the function returns. The values can be accessible before the compiling  of the program.  ***Some Disadvantages:***  1. Variables declared within a function have a limited scope and lifetime.  2. It is not directly accessible during the compilation process.  3. When the function returns the variables within this function is deallocated from the stack.  ***Heap Memory:***  Heap memory is a runtime memory allocation technique that allows access to function variables after the compilation of a program.It is a dynamic memory allocation technique that retains the values of function variables even after the function has completed its execution.  The values of the function variables are stored in the heap memory.  ***Advantages of heap memory:***  1.Variables declared within a function can be accessed anywhere in the program.  2. It is directly accessible during the execution process.  3. It requires additional memory for variable allocation. |

| Question No. 1-b |
| --- |
| Why do we need dynamic memory allocation? Explain with examples. |
| Answer No. 1-b |
| ***Why Dynamic Memory Allocation:*** Dynamic memory allocation is the process of storing values at run time of a program. As the static memory can’t hold the values of a function after the function returns we cannot use the variables or values inside this function from another function in the program.   | #include<bits/stdc++.h>  using *namespace* std;  *int*\* dynamic\_array(*int* *x*)  {  *int*\* arr = new *int*[x];  for (*int* i = 0; i < x; i++)  {  cin>>arr[i];  }  return arr;  }  *int* main()  {  *int* n;  cin>>n;  *int*\* arr = dynamic\_array(n);  for (*int* i = 0; i < n; i++)  {  cout<<arr[i]<<" ";  }  return 0;  } | #include<bits/stdc++.h>  using *namespace* std;  *int*\* array\_return(*int* *x*)  {  *int* arr[*x*];  for (*int* i = 0; i < *x*; i++)  {  cin>>arr[*x*];  }  return arr;  }  *int* main()  {  *int* n;  cin>>n;  *int* \*arr = array\_return(n);  for (*int* i = 0; i < n; i++)  {  cout<<arr[i]<<" ";  }  return 0;  } | | --- | --- |   Fig:Dynamic memory Fig:static memory  ***Example:***  For example if we want to return an array from a built-in function to the main function in the case of a static array in Fig(Static Memory) we see that the static memory first takes n size of memory then it calls array\_return function and passes value n. Then the static memory allocation is the size of an array. The loop takes n values int type in arr[n]. Then the function returns the address of the array arr and the main function receives an address.  When the loop is executed the function does not find anything on that address as the static memory deleted the function and the values of the function.  So it does not print the expected output.  Now we can overcome this problem by using Heap memory allocation by using a new operator. In Fig:Dynamic memory we see that an array of size x is created in heap memory and the address of the array is stored in static memory inside a pointer arr.  Then the for loop takes x input values in heap memory. Then the function returns the address of this array to the main function inside the pointer arr. So now the function is deleted and the stack memory is empty but another pointer is now holding the address of the array. The array does not get deleted.  So using the address we can now print the values inside the array as expected. |

| Question No. 1-c |
| --- |
| How to create a dynamic array? What are the benefits of it? |
| Answer No. 1-c |
| To create a dynamic array we need to use an operator called new.  The array is created inside the heap memory and the address of that array is stored in a pointer that points to that address at static memory.  Syntax:  data\_type\* array\_name = new data\_type[size];  #include<bits/stdc++.h>  using *namespace* std;  *int*\* dynamic\_array(*int* *x*)  {  *int*\* arr = new *int*[*x*];  for (*int* i = 0; i < *x*; i++)  {  cin>>arr[i];  }  return arr;  }  *int* main()  {  *int* n;  cin>>n;  *int*\* arr = dynamic\_array(n);  for (*int* i = 0; i < n; i++)  {  cout<<arr[i]<<" ";  }    return 0;  }  ***Benefits of Dynamic Array***  1.Access of a function's data is doable outside the declared function.  2.The values of the array can be manipulated after the compilation of the program means  at runtime.  3.Reusability of the values.  4.The array can be deleted after an operation.  5.Flexibility of determining the size of an array at runtime.  6.Memory efficiency as the heap memory allocates memory only as needed.  7.Dynamic arrays serve as the foundation for more complex data structures. |

| Question No. 2-a |
| --- |
| How does class and object work? How to declare an object? |
| Answer No. 2-a |
| ***Class***:A class is a blueprint that has one or more properties combined together to create a structure that describes what an object can do.  It defines the properties(data) and behaviors(functions) that the object of that class will have.  *class* Student  {  *public:*  *char* name[100];  *int* roll;  *int* cls;  *char* section;  };  *int* main()  {  Student Isti;  Isti.roll=29;  Isti.cls=9;  Isti.section='A';  *char* nm[100]="Isti";  strcpy(Isti.name,nm);  return 0;  }  For example a class is declared Student and the access modifier is public.  The class has four properties.  When an object Isti from the class Student is created now we can set the object's value for name,roll,cls and section sequentially.  ***Object***:An object is a specific instance of a class. It is created from the class and represents a unique entity with iu’s own set of data and behavior.  ***Syntax***  Class\_name object\_Name;  object\_Name.properties=value; |

| 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:*** A constructor is a special function within a class that is automatically called when an object of that class is created.  It is used to initialize the objects data members and perform any necessary setup or initialization task.  ***Example:***  *class* Student  {  *public:*  *char* name[100];  *int* roll;  *int* cls;  *char* section;  /\*constructor\*/  Student(*int* *r*,*int* *s*,*int* *c*,*char*\* *n*)  {  roll=*r*;  section=*s*;  cls=*c*;  strcpy(name,*n*);  }  };  *int* main()  {  Student Isti(29,'A',9,"Isti");  cout<<"Name: "<<Nasrullah.name<<endl<<"Roll: "<<Nasrullah.roll<<endl;  cout<<"Section: "<<Nasrullah.section<<endl<<"Class: "<<Nasrullah.cls<<endl;  ewturn 0;  }  A constructor Student is created as the same name of the class. An object is created (Student Isti()) in the main function and the values are passed to the constructor function 29,A,9,Isti sequentially. |

| 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 |
| #include<bits/stdc++.h>  using *namespace* std;  *class* Person  {  *public:*  *char* name[100];  *float* height;  *int* age;  Person(*char*\* *nam*,*float* *h*,*int* *a*)  {    strcpy(name,*nam*);  height=*h*;  age=*a*;  }  };  *int* main()  {  *char* name[100]="Isti";  Person\* Isti = new Person(name,5.9,24);  cout<<"Name : "<<Isti->name<<endl<<"Height : "<<Isti->height<<endl<<"Age : "<<Isti->age<<endl;  return 0;  } |

| Question No. 3-a |
| --- |
| What is the size that an object allocates to the memory? |
| Answer No. 3-a |
| The size of an object in memory is determined by the total memory required to store all it’s data members. The size of an object can vary depending on factors such as the data types of it’s members and any function present.  *class* Person  {  *public:*  *char* name[100];  *float* height;  *int* age;  Person(*char*\* *nam*,*float* *h*,*int* *a*)  {    strcpy(name,*nam*);  height=*h*;  age=*a*;  }  };  *int* main()  {  *char* name1[100]="Isti";  Person\* Isti = new Person(name1,5.9,29);  For instance in this case when a dynamic object Person() is created by using new operator the heap memory created total 108 byte(char(1\*100)+float(1\*4)+int(1\*4)) memory in heap memory. |

| Question No. 3-b |
| --- |
| Can you return a static object from a function? If yes, show with an example. |
| Answer No. 3-b |
| Yes , it is possible to return a static object from a function in C++.  A static object has a lifetime that extends throughout the entire program execution , and its storage duration is static. This means that the object is created once and retains its value across multiple function calls.  #include<bits/stdc++.h>  using *namespace* std;  *class* Person  {  *public:*  *int* age;  Person(*int* *a*)  {  age=*a*;  }  *void* displayValue()  {  cout<<age;  }  };  Person& Staticobject()  {  static Person age(24);  return age;  }  *int* main()  {  Person& objRef = Staticobject();  objRef.displayValue();  return 0;  }  In this example a function is created Staticobject() and the class is person that holds the address that is returned by this function. A static object is created age and the address of age is sent to the function. Then the main function calls to displayValue() function.  And the displayvalue function displays the expected output. |

| Question No. 3-c |
| --- |
| Why do we need -> (arrow sign)? |
| Answer No. 3-c |
| The arrow sign refers to a dynamic object address. We can access a dynamic object parameter or assign a value to that variable.  When we print a static object we use object\_name.object\_parameter .  Now to print a dynamic object parameter we use a pointer to access the parameter value.  (\*object\_name).object\_parameter .  We can simply replace the first parentheses ,pointer sign and dor by this shortcut technique -> arrow sign.  This makes the use of dynamic objects easy.  This arrow sign only works if the object is a dynamic object.  #include<bits/stdc++.h>  using *namespace* std;  *class* Person  {  *public:*  *char* name[100];  *float* height;  *int* age;  Person(*char*\* *nam*,*float* *h*,*int* *a*)  {    strcpy(name,*nam*);  height=*h*;  age=*a*;  }  };  *int* main()  {  *char* name1[100]="Isti";  Person\* Isti = new Person(name1,5.9,29);  *char* name2[100]="Nasrullah";  Person\* Nasrullah = new Person(name2,6.00,10);  if(Nasrullah->age > Isti->age)  {  cout<<Nasrullah->name<<endl;  }  else{  cout<<Isti->name<<endl;  }  return 0;  }  As we can see the age is compared directly by using the pointer that is  if(Nasrullah->age > Isti->age)  If the age at the location \*Nasrullah is greater then it will print Nasrullah.  In this case the output matches the expectation. |

| 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 |
| #include<bits/stdc++.h>  using *namespace* std;  *class* Person  {  *public:*  *char* name[100];  *float* height;  *int* age;  Person(*char*\* *nam*,*float* *h*,*int* *a*)  {    strcpy(name,*nam*);  height=*h*;  age=*a*;  }  };  *int* main()  {  *char* name1[100]="Isti";  Person\* Isti = new Person(name1,5.9,29);  *char* name2[100]="Nasrullah";  Person\* Nasrullah = new Person(name2,6.00,25);  if(Nasrullah->age > Isti->age)  {  cout<<Nasrullah->name<<endl;  }  else{  cout<<Isti->name<<endl;  }  return 0;  } |