Ans 1. a) Highlight the merits of Jackson structured development methodology.(2)

* It is designed to solve the real-time problems.
* JSD modelling focuses on time.
* It considers simultaneous processing and timing.
* Provides functionality in the real world.
* It is a better approach for microcode applications.

b. When is it required to create an abstract class? Explain it with suitable example.( **1+1)**

The purpose of an **abstract class**  is to provide an appropriate base class from which other classes can inherit. Abstract classes cannot be used to instantiate objects and serves only as an **interface**. Attempting to instantiate an object of an abstract class causes a compilation error.

#include<iostream>

using namespace std;

class Base

{

int x;

public:

virtual void fun() = 0;

int getX() { return x; }

};

// This class inherits from Base and implements fun()

class Derived: public Base

{

int y;

public:

void fun() { cout << "fun() called"; }

};

int main(void)

{

Derived d;

d.fun();

return 0;

}

c. “Inline functions are different from Macros.” Justify (2)

“ Yes, inline functions are different from Macros.”

An inline function is a normal function that is defined by the **inline** keyword. An inline function is a short function that is expanded by the compiler. And its arguments are evaluated only once. An inline functions are the short length functions that are automatically made the inline functions without using the **inline** keyword inside the class whereas, Macros is also called **preprocessors directive**. The macros are defined by the **#define** keyword. Before the program compilation, the preprocessor examines the program whenever the preprocessor detects the macros then preprocessor replaces the macro by the macro definition.

An inline function is defined by the **inline** keyword whereas the macros are defined by the **#define** keyword.

Inline function is terminated by the curly brace at the end while the macro is not terminated by any symbol, it is terminated by a new line.

In C++, inline may be defined either inside the class or outside the class whereas the macro is all the time defined at the beginning of the program.

d. In what order the class constructors are called when a derived class object is created? **(2)**

Whenever we create an object of a class, the default constructor of that class is invoked automatically to initialize the members of the class.

If we inherit a class from another class and create an object of the derived class, it is clear that the default constructor of the derived class will be invoked but before that the default constructor of all of the base classes will be invoke, i.**e the order of invocation is that the base class’s default constructor will be invoked first and then the derived class’s default constructor will be invoked.**

e.

|  |
| --- |
| Define constant pointer. What is the usage of pointers in C++? (1+1) |

In constant pointers, the pointer points to a fixed memory location and the value at that location can be changed because it is a variable, but the pointer **will always point to the same location** because it is made constant here.

**Usage of pointers:**

1. To pass arguments by reference
2. For accessing array elements
3. To return multiple values
4. Dynamic memory allocation
5. To implement data structures
6. To do system-level programming where memory addresses are useful.

Ans 2 a. **“Jackson Structured Development aims at developing a maintainable software”.**

Justify above statement and explain all the phases of JSD. (5)

**Jackson System Development (JSD)** is a method of system development that covers the software life cycle either directly or by providing a framework into which more specialized techniques can fit. JSD can start from the stage in a project when there is only a general statement of requirements. However many projects that have used JSD actually started slightly later in the life cycle, doing the first steps largely from existing documents rather than directly with the users.

**JSD, Jackson Structure Design is a methodology to specify and design systems in which time factor is significant and system may be described using sequence of events**. Developed by Michael A. Jackson , this design method considers the fact that the design of the system is an extension of the programme design. The **purpose of this design method is to create a maintainable software**. The method addresses all stages of the software development life cycle. It has three phases.

### ****Phases of JSD:****

JSD has 3 phases:

1. **Modelling Phase:** In the modelling phase of JSD, the designer creates a collection of entity structure diagrams and identifies the entities in the system, the actions they perform, the attributes of the actions and the time order of the actions in the life of the entities.
2. **Specification Phase:** This phase focuses on actually what is to be done? Previous phase provides the basic for this phase. An sufficient model of a time-ordered world must itself be time-ordered. Major goal is to map progress in the real world on progress in the system that models it.

**Implementation Phase:** In the implementation phase JSD determines how to obtain the required functionality. Implementation way of the system is based on the transformation of the specification into an efficient set of processes. The processes involved in it should be designed in such a manner that it would be possible to run them on available software and hardware.

Ans 2 b. How can we map object-oriented concepts using non-object-oriented language?(5)

Implementing an object-oriented design in a non-object-oriented language requires basically the same steps as implementing a design in an object-oriented language. The programmer using a non-object-oriented language must map object-oriented concepts into the target language, whereas the compiler for an object-oriented language performs such: napping automatically. The steps required to implement a design are:

* **Translate classes into data structures**

Each class is implemented as a single contiguous block of attributes. Each attribute contains a variable. Now an object has state and identity and is subject to side effects.

A variable that identifies an object must therefore be implemented as a sharable reference.

* **Pass arguments to methods**

Every method has at least one argument. In a non-object-oriented language, the argument must be made explicit.

Methods can contain additional objects as arguments. In passing an object as an argument to a method, a reference to the object must be passed if the value of the object can be updated within the method.

* Allocate storage for objects
* Implement inheritance in data structures
* Implement method resolution
* Implement associations
* Deal with concurrency
* Encapsulate internal details of classes

|  |
| --- |
| **3 a)**  **Write the output of the : (5)**  #include <iostream>  using namespace std;  class A  { int id;  static int count;  public:  A() {  count++;  id = count;  cout << "constructor for id " << id << endl; }  ~A() {  cout << "destructor for id " << id << endl; }};  int A::count = 0;  int main() {  A a[3];  return 0; }  **Output:**  **Constructor for id 3**  **Destructor for id 3**  **Destructor for id 2**  **Destructor for id 1** |
| 3 b) Write the Output of: (5)  #include <stdio.h>  int main()  {  int i=3;  switch(i)  {  case 0: printf("Purple");  break;  case 1+1: printf("Blue");  break;  case 7/2: printf("Yellow");  break;  case 3%2: printf("Black");  break;  }  return 0;  }    Output :  Yellow |

**Q 4a) “A copy constructor is a member function that initializes an object using another object of the same class”.(5)**

Write a program to justify the above statement by implementing copy constructor.

Copy Constructor is used to declare and initialize an object from another object. These are special type of Constructors which takes an object as argument, and is used to copy values of data members of one object into other object.

Eg: abc c2(c1);

would define the object c2 and at the same time initialize it to the value of c1.

The process of initializing through a copy constructor is known as *copy initialization.*

**Example:**

class abc

{

int a, b;

public:

abc(int x, int y)

{

a = x;

b = y;

}

abc::abc(abc &p)

{

a = p.a;

b = p.b;

}

void showdata()

{

cout << a << " " << b << endl;

}

};

int main()

{

abc c1(10, 20);

abc c2(c1);

c1.showdata();

c2.showdata();

getch();}

**4b)Write a program to explain ‘this’ pointer in C++. (5)**

In C++ programming, **‘this’** is a keyword that refers to the current instance of the class. There can be 3 main usage of this keyword in C++.

It can be used to pass current object as a parameter to another method.

It can be used to refer current class instance variable.

It can be used to declare indexers.

**A program to explain ‘this’ pointer.**

**#include <iostream>**

**using namespace std;**

**class Employee {**

**public:**

**int id; //data member (also instance variable)**

**string name; //data member(also instance variable)**

**float salary;**

**Employee(int id, string name, float salary)**

**{**

**this->id = id;**

**this->name = name;**

**this->salary = salary;**

**}**

**void display()**

**{**

**cout<<id<<"  "<<name<<"  "<<salary<<endl;**

**}**

**};**

**int main(void) {**

**Employee e1 =Employee(101, "Sonoo", 890000); //creating an object of Employee**

**Employee e2=Employee(102, "Nakul", 59000); //creating an object of Employee**

**e1.display();**

**e2.display();**

**return 0;**

**}**

**Output:**

**101 Sonoo 890000**

**102 Nakul 5900**

**Q5 a) Compare SA/SD and Jackson Structured Development? Is JSD methodology Object Oriented? Give proper justification? (5+5)**

**Difference Between SA/SD and Jackson Structured Development** 

|  |  |
| --- | --- |
| **SA/SD** | **Jackson Structured Development** |
|  |  |
|  |  |
| The main focus is on the process and procedures of the system. | The main focus is on data structure and real-world objects that are important. |
| It uses System Development Life Cycle (SDLC) methodology for different purposes like planning, analyzing, designing, implementing, and supporting an information system. | It uses Incremental or Iterative methodology to refine and extend our design. |
| It is suitable for well-defined projects with stable user requirements. | It is suitable for large projects with changing user requirements. |
| Risk while using this analysis technique is high and reusability is also low. | Risk while using this analysis technique is low and reusability is also high. |
| Structuring requirements include DFDs  (Data Flow Diagram), Structured Analysis, ER (Entity Relationship) diagram, CFD (Control Flow Diagram), Data Dictionary, Decision table/tree, and the State transition diagram. | Requirement engineering includes the Use case model (find Use cases, Flow of events, Activity Diagram), the Object model (find Classes and class relations, Object interaction, Object to ER mapping), Statechart Diagram, and deployment diagram. |
| This technique is old and is not preferred usually. | This technique is new and is mostly preferred. |

**JSD is an object-based method of analysis. A JSD specification can be seamlessly implemented using an object-oriented programming language: entities, actions, and attributes of a JSD specification map into objects, methods and instance variables of an OOP language. JSD is object-based, that is, JSD models the behavior of the objects of interest in a user's problem domain and these factors make JSD an object oriented methodology.**

**Q5b)**

**“After analysis phase, we have the object, dynamic, and functional models but the object model is the main framework around which the design is constructed”.**

Justify it with example? (10)

In the system analysis or object-oriented analysis phase of software development, the system requirements are determined, the classes are identified and the relationships among classes are identified.

The three analysis techniques that are used in conjunction with each other for object-oriented analysis are object modelling, dynamic modelling, and functional modelling.

**Object Modelling**

Object modelling develops the static structure of the software system in terms of objects. It identifies the objects, the classes into which the objects can be grouped into and the relationships between the objects. It also identifies the main attributes and operations that characterize each class.

The process of object modelling can be visualized in the following steps −

* Identify objects and group into classes
* Identify the relationships among classes
* Create user object model diagram
* Define user object attributes
* Define the operations that should be performed on the classes
* Review glossary

**Dynamic Modelling**

After the static behavior of the system is analyzed, its behavior with respect to time and external changes needs to be examined. This is the purpose of dynamic modelling.

Dynamic Modelling can be defined as “a way of describing how an individual object responds to events, either internal events triggered by other objects, or external events triggered by the outside world”.

The process of dynamic modelling can be visualized in the following steps −

* Identify states of each object
* Identify events and analyze the applicability of actions
* Construct dynamic model diagram, comprising of state transition diagrams
* Express each state in terms of object attributes
* Validate the state–transition diagrams drawn

**Functional Modelling**

Functional Modelling is the final component of object-oriented analysis. The functional model shows the processes that are performed within an object and how the data changes as it moves between methods. It specifies the meaning of the operations of object modelling and the actions of dynamic modelling. The functional model corresponds to the data flow diagram of traditional structured analysis.

The process of functional modelling can be visualized in the following steps −

* Identify all the inputs and outputs
* Construct data flow diagrams showing functional dependencies
* State the purpose of each function
* Identify constraints
* Specify optimization criteria

Out of all the three modelling techniques object modelling is the main technique around which the design constraints revolve because it is the Object modelling which develops the static structure of the software system in terms of objects. It identifies the objects, the classes into which the objects can be grouped into and the relationships between the objects. It also identifies the main attributes and operations that characterize each class.

**Q6a) How function overloading is different from overriding. Write a program to overload + operator using friend function to concatenate two strings. (5+5)**

Function overloading and overriding play similar roles. They both provide an abstraction over the interface so that the end user doesn’t have to think much about the context and pass in the arguments. However, there are subtle differences between the two approaches.

| **Function Overloading** | **Function Overriding** |
| --- | --- |
| Function overloading can be used in normal functions as well as in classes (e.g., constructor overloading is a classic example where you would vary the number/type of arguments for different initializations). | Function overriding applies exclusively to an inherited class (or in other words a subclass). |
| Function overloading is resolved at compile time. | Function overriding is resolved at run time. |
| Overloaded functions are in the same scope. | Overridden functions are in different scopes. |
| Overloaded functions have different function signatures. | Overridden functions have the same function signatures. |

**/ C++ Program to concatenate two strings using friend function and operator overloading**

**#include <iostream>**

**#include <string.h>**

**using namespace std;**

**// Class to implement operator overloading function**

**// for concatenating the strings**

**class AddString {**

**public:**

**// Class object of string**

**char str[100];**

**// No Parameter Constructor**

**AddString() {}**

**// Parameterized constructor to**

**// initialize class Variable**

**AddString(char str[])**

**{**

**strcpy(this->str, str);**

**}**

**friend Addstring operator+(Addstring a1,Addstring a2);**

**// Overload Operator+ to concatenate the strings**

**AddString operator+(AddString &S2)**

**{**

**// Object to return the copy**

**// of concatenation**

**AddString S3;**

**// Use strcat() to concat two specified string**

**strcat(this->str, S2.str);**

**// Copy the string to string to be return**

**strcpy(S3.str, this->str);**

**// return the object**

**return S3; }**

**};**

**// Driver Code**

**int main()**

**{**

**// Declaring two strings**

**char str1[] = “Information";**

**char str2[] = “Technology";**

**// Declaring and initializing the class**

**// with above two strings**

**AddString a1(str1);**

**AddString a2(str2);**

**AddString a3;**

**// Call the operator function**

**a3 = a1 + a2;**

**cout << "Concatenation: " << a3.str;**

**return 0; }**

**}**

**Ans 6b)** Differentiate between call by value and call by reference. Write a program to support the explanation (5+5)

**Call by value**

In call by value, **original value is not modified,** value being passed to the function is locally stored by the function parameter in stack memory location. If you change the value of function parameter, it is changed for the current function only. It will not change the value of variable inside the caller method such as main().

#include <iostream>

using namespace std;

void change(int data);

int main()

{

int data = 3;

change(data);

cout << "Value of the data is: " << data<< endl;

return 0;

}

void change(int data)

{

data = 5;

}

Output:

Value of the data is: 3

**Call by reference**

In call by reference, original value is modified because we pass reference (address).Here, address of the value is passed in the function, so actual and formal arguments share the same address space. Hence, value changed inside the function, is reflected inside as well as outside the function

#include<iostream>

using namespace std;

void swap(int \*x, int \*y)

{

 int swap;

 swap=\*x;

 \*x=\*y;

 \*y=swap;

}

int main()

{

 int x=500, y=100;

 swap(&x, &y);  // passing value to function

 cout<<"Value of x is: "<<x<<endl;

 cout<<"Value of y is: "<<y<<endl;

 return 0;

}

Output:

Value of x is: 100

Value of y is: 500

**Ans7a)** How virtual functions are used to support "run time polymorphism”? Justify your answer with proper implementation in C++. (5+5)

A **virtual function** is a *member function* that is declared in the *base* class using the keyword **virtual** and is re-defined (Overridden) in the *derived* class. It tells the compiler to perform late binding where the compiler matches the object with the right called function and executes it during the runtime. This technique of falls under Runtime Polymorphism.

The term**Polymorphism** means the ability to take many forms. It occurs if there is a hierarchy of classes that are all related to each other by *inheritance*. In simple words, when we break down Polymorphism into*‘Poly – Many’*and*‘morphism – Forms’* it means showing different characteristics in different situations.

A virtual function is another way of implementing run-time polymorphism in C++. It is a special function defined in a base class and redefined in the derived class. To declare a virtual function, you should use the virtual keyword. The keyword should precede the declaration of the function in the base class.

If a virtual function class is inherited, the virtual class redefines the virtual function to suit its needs.

If a virtual function class is inherited, the virtual class redefines the virtual function to suit its needs.

#include <iostream>

using namespace std;

class ClassA {

public:

virtual void show() {

cout << "The show() function in base class invoked..." << endl;

}

};

class ClassB :public ClassA {

public:

void show() {

cout << "The show() function in derived class invoked...";

}

};

int main() {

ClassA\* a;

ClassB b;

a = &b;

a->show();

}

**Ans 7b)**

How can we access a private member of another class within the program? Implement it by writing the code in C++. (5+5)

A **friend class** can access private and protected members of other class in which it is declared as friend. It is sometimes useful to allow a particular class to access private members of other class. **For example**, a LinkedList class may be allowed to access private members of Node. A friend class can access both private and protected members of the class in which it has been declared as friend.

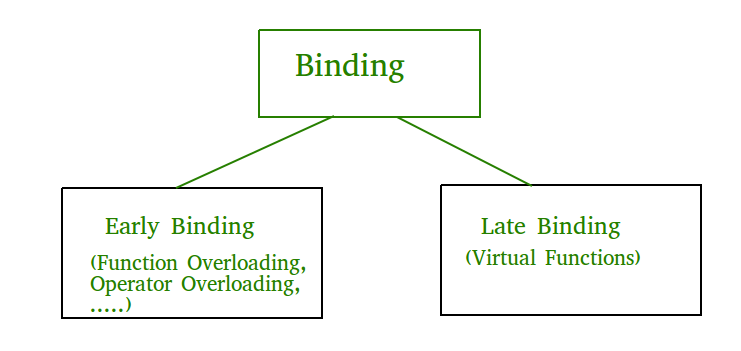
|  |
| --- |
| #include<iostream>  using namespace std;  class A  {  int x;  public:      A()      {          x=10;      }      friend class B;    //friend class  };  class B  {   public:          void display(A &t)          {              cout<<endl<<"The value of x="<<t.x;          }  };  main()  {      A \_a;      B \_b;     \_b.display(\_a);      return 0;  } |

**Output:**

The value of x=10

In this example, class B is declared as a friend inside the class A. Therefore, B is a friend of class A. Class B can access the private members of class A.

**Ans8a)** Which technique is used to implement early and late binding in polymorphism? Implement it by writing the code in C++ to support your answer.(5+5)



Binding refers to the process of converting identifiers (such as variable and performance names) into addresses. Binding is done for each variable and functions. For functions, it means that matching the call with the right function definition by the compiler. It takes place either at compile time or at runtime.

**Early Binding (compile-time time polymorphism):** As the name indicates, compiler (or linker) directly associates an address to the function call. It replaces the call with a machine language instruction that tells the mainframe to leap to the address of the function. By default early binding happens in C++.

|  |
| --- |
| #include<iostream>  using namespace std;  class Base  {  public:      void show() { cout<<" In Base \n"; }  };  class Derived: public Base  {  public:      void show() { cout<<"In Derived \n"; }  };  int main(void)  {      Base \*bp = new Derived;      // The function call decided at      // compile time (compiler sees type      // of pointer and calls base class      // function.      bp->show();      return 0;} |

**Output:**

In Base

**Late Binding : (Run time polymorphism)** In this, the compiler adds code that identifies the kind of object at runtime then matches the call with the right function definition This can be achieved by declaring a virtual function.

|  |
| --- |
| #include<iostream>  using namespace std;  class Base  {  public:      virtual void show() { cout<<" In Base \n"; }  };  class Derived: public Base  {  public:    void show() { cout<<"In Derived \n"; }  };  int main(void)  {      Base \*bp = new Derived;      bp->show();  // RUN-TIME POLYMORPHISM      return 0;  } |

**Output:**

In Derived

|  |
| --- |
| **Ans 8b) “Inheritance is a feature or a process in which, new classes are created from the existing classes”.** With reference to the above statement explain the significance of inheritance. Implement the concept of multilevel inheritance with a suitable example.  **Significance of Inheritance**  We use inheritance in C++ for the reusability of code from the existing class. C++ strongly supports the concept of reusability. Reusability is yet another essential feature of OOP(Object Oriented Programming).It is always good to reuse something that already exists rather than trying to create the one that is already present, as it saves time and increases reliability. We use inheritance in C++ when both the classes in the program have the same logical domain and when we want the class to use the properties of its superclass along with its properties.  For example, there is a base class or parent class named “Animal,” and there is a child class named “Dog,” so, here dog is an animal, so in “Dog class,” all the common properties of the “Animal” class should be there, along with its property of dog animal  **Multilevel Inheritance.**  Derivation of a class from another derived class is called Multilevel Inheritance. Class A is the base class for the derived class B, which in turn serves as a base class for the derived class C. Class B provides a link for the inheritance between A and C and is known as an intermediate base class. The chain A, B, C is known as the **inheritance path**.  Multilevel Inheritance  // C++ program to implement  // multilevel inheritance  #include<iostream>  using namespace std;  // Base class A  class A  {      public:          A()          {              int a = 5, b = 6, c;              c = a + b;              cout << "Sum is:" <<                       c << endl;          }  };  // Class B  class B: public A  {      public:          B()          {              int d = 50,e = 35, f;              f = d - e;              cout << "Difference is:" <<                       f << endl;          }  };  // Derived class C  class C: public B  {      public:          C()          {              int g = 10, h = 20, i;              i = g \* h;              cout << "Product is:" <<                       i << endl;          }  };  // Driver code  int main()  {  C obj;  return 0;  } |

**Output**

Sum is:11

Difference is:15

Product is:200

**Ans 9a)**

#include<iostream>

using namespace std;

class dist

{

public:

int feet,inch,x,y,z;

void input()

{

cout<<"enter feet and inches:"<<"\n";

cin>>feet>>inch;

}

void display\_Distance()

{

cout<<"The distance is ";

cout<<feet<<"feet"<<inch<<"inch"<<"\n";

}

void add(dist x,dist y)

{

feet=x.feet+y.feet;

inch=x.inch+y.inch;

if(inch>=12)

{

feet=feet+1;

inch=inch-12;

}

}

};

int main()

{

dist x,y,z;

x.input();

y.input();

z.add(x,y);

z.display\_Distance();

}

**Ans 9b)**

#include<iostream>

#include<math.h>

using namespace std;

class account

{

public:

char cust\_name[50],type;

long acc\_no,balance;

void get()

{

cout<<"\nenter customer name: ";

cin>>cust\_name;

cout<<"enter account number:";

cin>>acc\_no;

cout<<"enter types s for saving and c for current:";

cin>>type;

cout<<"enter balance:";

cin>>balance;

}

void display()

{

cout<<"\n Customer Name: "<<cust\_name;

cout<<"\n Account Number: "<<acc\_no;

cout<<"\n Type: "<<type;

cout<<"\n Balance: "<<balance;

}

void deposit()

{

int amt;

cout<<"\nenter the amount which you want to deposit:";

cin>>amt;

balance+=amt;

}

};

class sav\_acct:public account

{

public:

int intr;

void comp\_intrest()

{

int t1,r1=10;

intr=balance\*pow(1+r1/12,t1);

cout<<intr;

balance+=intr;

}

void withdraw()

{

int amt;

cout<<"\nenter the amount which do you want to withdraw:";

cin>>amt;

if(balance>amt)

balance-=amt;

else

cout<<"\namount can't be withdrawn due to insufficient balance";

}

};

class curr\_acct:public account

{

public:

void min\_bal()

{

if(balance<500)

{

cout<<"\npenality imposed: \nnew balance is:"<<balance-50;

}

else

{

cout<<"\nno penalty imposed";

}

}

void withdraw()

{

int amt;

cout<<"\nenter amount to be withdrawn:";

cin>>amt;

if(amt>balance)

{

cout<<"amount can't be withdrawn due to insufficient balance";

}

else

{

balance-=amt;

}

}

};

int main()

{

sav\_acct s1;

curr\_acct c1;

c1.get();

c1.display();

c1.deposit();

c1.display();

c1.withdraw();

c1.display();

c1.min\_bal();

s1.get();

s1.display();

s1.deposit();

s1.comp\_intrest();

s1.display();

s1.withdraw();

s1.display();

}