CHHATTISGARH SWAMI VIVEKANAND TECHNICAL UNIVERSITY

छत्तीसगढ़ स्वामी विवेकानंद तकनीकी विश्वविद्यालय

To be filled, scanned and kept at 1st page of Answer Booklet.

Nov-Dec 2021 Examination

Student Name: V. Om Sai Ng Nageshwar Sharma

Mobile No.: 8602727389

Email ID: Nageshwar.sharma@gmail.com

Enrollment No.: | B | J | 4 | 5 | 9 | 9 |

Roll No.: | 3 | 0 | 3 | 3 | 0 | 2 | 2 | 2 | 0 | 0 | 2 | 0 |

Course: B.Tech Semester: 3rd

Branch/Specialization: CSE

Subject Code: | B | 0 | 2 | 2 | 3 | 1 | 3 | (| 0 | 2 | 2 |) |

Subject Name: Principles of Programming languages

Regular/Backlog: Regular

Date of Exam: 25 / 03 /2022

Note:

1) Only above format is to be used for Nov-Dec 2021 Exams. Older/earlier format will not be accepted.
2) Nomenclature to be mentioned in the Answer Booklet should be Subject code_Roll No. only.
3) Only Roll No. generated in Admit Card must be filled (College Transfer students must take care in filling their Roll Nos,).

I certify that above information given there in is correct and I shall be personally responsible for the same if proved wrong/false later on.

Signature:

CHHATTISGARH SWAMI VIVEKANAND TECHNICAL UNIVERSITY

Bhilai, Durg, Chhattisgarh

Admit Card for Session Nov-Dec 2021

STUDENT DETAILS

Registration No.	BJ4599	Roll No.	303302220029
College Name	033 - SHRI SHANKARACHARYA INSTITUTE OF PROFESSIONAL MANAGEMENT & TECHNOLOGY,RAIPUR		
Student's Name	V OM SAI NAGESHWAR SHARMA		
Gender	Male	Date of Birth	17-02-2001
Father's Name	V Someshkhar Sharma	Current Semester	3 SEMESTER
Program	B.Tech Computer Science Engineering	Course	B.Tech

SUBJECT DETAILS

Sl.	Semester	Subject Type	Subject Code	Subject Name	Exam Session	Exam Type	ToAppear	Date & Time of Exam
1	3 SEMESTER	Sessional	B000306(049)	Personality Development	Nov-Dec 2021	Regular	Y	
2	3 SEMESTER	Theory	B000311(022)	Data structure & Algorithms	Nov-Dec 2021	Regular	Y	23/03/22 10:00 am
3	3 SEMESTER	Theory	B000312(014)	Mathematics - III	Nov-Dec 2021	Regular	Y	21/03/22 10:00 am
4	3 SEMESTER	Theory	B000313(022)	Principles of Programming Languages	Nov-Dec 2021	Regular	Y	25/03/22 10:00 am
5	3 SEMESTER	Theory	B000314(022)	Digital Electronics& Logic Design	Nov-Dec 2021	Regular	Y	
6	3 SEMESTER	Theory	B000315(022)	Operating Systems	Nov-Dec 2021	Regular	Y	
7	3 SEMESTER	Practical	B000321(022)	Data structure & Algorithms Laboratory	Nov-Dec 2021	Regular	Y	
8	3 SEMESTER	Practical	B000322(022)	Digital Electronics& Logic Design Laboratory	Nov-Dec 2021	Regular	Y	
9	3 SEMESTER	Practical	B000323(022)	Operating Systems Laboratory (UNIX)	Nov-Dec 2021	Regular	Y	
10	3 SEMESTER	Practical	B000324(022)	Software Laboratory (Gc Lab/MATLAB)	Nov-Dec 2021	Regular	Y	

Signature should match with uploaded signature of candidate

Signature of the Candidate (while receiving) | Signature of the Principal | Signature of COE

INSTRUCTION FOR WRITTEN EXAMINATION

1. Candidates suffering from any diseases which would render their presence in the Examination Hall undesirable in the interests of other candidates will not be allowed to enter the Examination Hall. In exceptional cases the Centre Superintendent informed by a candidate in advance may make special arrangements.
2. The doors of the Examination Hall will be opened half an hour before the examination starts on the first date and 15 minutes before on other days. Candidates are required to sit in their allotted seats 5 minutes before the examination starts. Candidates arriving after the start of examination will not be admitted unless specially permitted by the Centre Superintendent. The Candidate shall not be allowed to consult any book or paper either in the Examination Hall or outside.
3. Candidates should bring their own pens and Mathematical instruments. Candidates should not be in possession of printed manuscripts (Other than their Admit Card), books and other documents while in the Examination Hall. Mutual consultation in the Examination Hall, bringing in Electronic equipments and other printed manuscripts from outside is forbidden.
4. Candidates are not allowed to consult any book or paper either in the Examination Hall or outside.
5. Candidates are not allowed to leave the hall until an hour after an examination starts. They should not leave their seats until they have submitted their answer books to an invigilator. Candidates should not tear off leaf behind on the desks. It is the duty of the candidate to see that his answer paper is received by the invigilator. No candidate will be allowed to re-enter the examination hall after once quitting it on submission of his answer paper. In cases of urgent need however candidate may be permitted by the senior invigilator to leave the hall temporarily and can move than once in an hour (restal super-vision. The period of temporary absentia should not exceed 3 minutes.
6. Candidates are not permitted to talk to each other in the Examination Hall. No one should receive help from or lender another in any manner.
7. Candidates should write their Regd No and Roll No on the cover page of mother answer book. Candidates are forbidden to write their own name and Code No of his/her college in the answer book while answering the questions. Giving a false Roll No. is serious offence and no answer book without the candidates Roll Number clearly written on the cover page will be examined.
8. Candidates should not, while in the examination hall write anything on the question paper or blotting papers nor should they take out of the hall any paper of nor than the question paper They should not also send the question paper for the examination outside thrhout the duration of the examination.
9. If a candidate is found with any paper not connected with the examination as mentioned in 4 he shall have to surrender it to the invigilator on demand and make a written statement. A necessary refusal to surrender the material or to make a statement shall render the candidate liable to instant expulsion from the Centre of Examination.

Unit - 1

Ans(a)

Module: A module is a software system component or part of a program that contains one or more routines.

Modularization Criteria:

Modularization is the process of breaking a software system into a set of collaborating components. Each of these components should ideally have high cohesion and low coupling

Modularization is inherently a recursive process. A real world example of modularization would be a car.

A car is composed of an engine, doors, chassis, etc. However, each component is then composed of modules, i.e. the door has a window, door lock, handle, etc.

High cohession means that each of a components. Components are closely related to each other, i.e. above the door's components of the window, door lock, and handle are all closely related.

Low coupling means that each component should be independent of the other components. In the case of a car, the engine is clearly independent of the door.

Ans (c)

Pseudocode: It is an artificial and informal language that helps programmers develop algorithms. Pseudocode is a "text-based" detail (algorithmic) design tool.

The rules of pseudo-code are reasonably straigtforward. All statements showing "dependency" are to be indented. These include while, do, for, if, switch. Examples below will illustrate this notion.

Examples:

1.) If student's grade is greater than or equal to 60.

```
Print "passed"
else
Print "failed
```

2.) Set total to zero

Set grade counter to one

while grade counter is less than or equal to ten.

Input the next grade

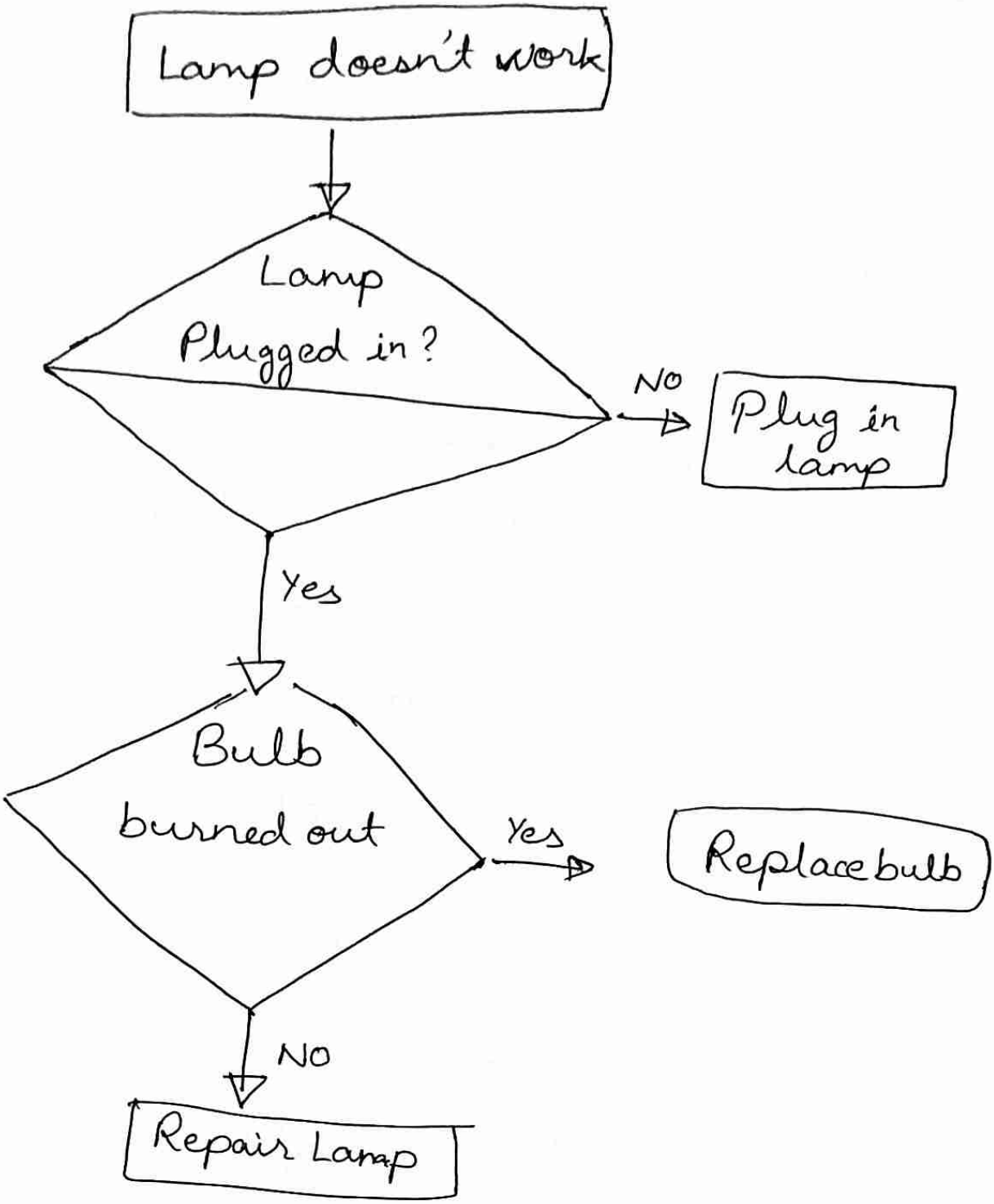
Add the grade into the total.

Set the class average to the total divided by ten.

Print the class average.

Flowchart :

A flowchart is a type of ~~dig~~ diagram that represents a workflow or process. A flowchart can also be defined as a diagramatic representation of an algorithm, a step-by-step approach to solving a task.



A simple flowchart representing a process for dealing with a non-functioning lamp.

Ans(b)

Step-wise refinement is a technique for software development using a top-down structured approach to solving a problem. It allows the developer to use a controlled method of developing an algorithm that will eventually solve a given problem.

Basically, step-wise refinement involves the use of a generated approach to a problem, perhaps written in pseudo-code. The next step would be to fill in one of the holes in the code.

The pseudo code describes a general, non-specific, ~~doing~~ way to solve the problem deffering the actual situations until last minute.

Then each step involves the use of refining (adding the logic) a step in the development of the algorithm to make it more specific, doing so one step at a time.

when everything has been refined we have a complete program, however, it takes multiple steps (step-wise refining) to get to that point where a complete, syntactically correct program emerges.

"Step-wis refinement" ultimately represents a "divide and ~~en~~ conquer" approach to design In other words, break a complex object into smaller, more managable pieces that can be reviewed and inspected before moving to the next level of detail.

Level 1: representing the overall product to be built.

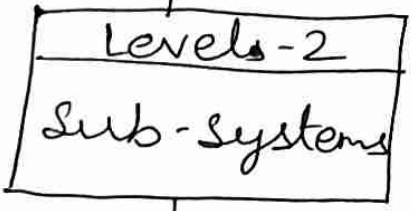
Level 2: Sub systems: representing the buisiness associated with the system (one or more).

Level 3: Procedures - representing the work flow of each sub-system. These are essential two types of procedures, Administrative - representing procedures executed by humans & computer.

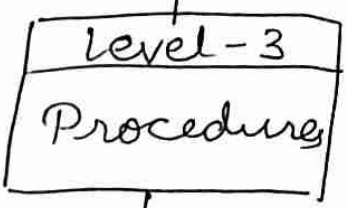
Level 4: Programs: representing the program needed to execute each computer procedure.

Under "Step-wise refinement" the levels are decomposed top-down during the design process, and implemented bottom-up.

Feasibity Study
- Information Recruitements
- Data Definitions.



System Designs
- Logical DB Design
- Illustrative inputs/output



Sub-systems Design.
- Workflow.
- Human/Machine interface
- Primary Physical files (including Manual files).

Software Engineering

- Program dependencies
- command language
- Program Logic.
- Method of Implementation
- Physical Input/Output File Layouts.

Unit - II (2)

Ans 2(a)

Characteristics of a programming Languages :-

1.) Readability → A good high level language will allow programs to be written in some way that resemble a quite-english description of the underlying algorithms. If care is taken, the coding may done in a way that is essentially self-documenting.

2.) Portability: High-level languages, being essentially machine independent, should be able to develop portable software.

3.) Familiar notation → A languages should have familier notation, so it can be understood by most of the programmers.

4) Widely available: language should be widely available and it should be possible to provide available and it should be possible to provide transistors for all the major machines and for all the major operating system.

Answer2(C)

~~Ans~~ ⟹ Different language evolution criteria are:

Readibility :-

Software development was largely thought of in term of writing code "LOC". Language constructs were designed more from the point of view of the computer than the user because of maintainence is determined in large part by the readability of programs, readability become an important measure of the

quality of programs and programming languages. The result is a crossover from Focus on machine orientation of focus on human orientation.

Write ability :-

It is a measure of how easily a language can be used to create programs for a chosen problem domain Most of the language characteristics that affect ~~readability~~ readability also affect write ability.

Simplicity :

A small number of primitive constructs and a consistent set of rules for combining them is much better than simply having a large number of primitives.

Expressivity:

It mean that a language has relatively convenient ways of specifying computations.

Ex → ++count ⇔ count = count + 1 // more convinient.

Support for abstraction :-

Abstraction means the ability to define and then use complicated structures or operations in ways that allow many of the details to be ignored. A process abstraction is the use of a subprogram to implement a sort algorithm that is seq required several times in a program instead of replicating it in all places where it is needed.

Exception handling:-

The ability to intercept run-time errors, take corrective measures, and then continue is a great aid to ~~relatively~~ reliability.

Aliansing:

It is having two or more distinct referencing methods, or names for the same memory cell. It is now widely the accepted that aliansing is a dangerous feature is a languages.

Answer 2(d)

Early Binding (Static Binding):

When perform Early Binding, an object is assigned to a variable to be of a specific object type. Early binding objects are basically a strong type objects or static type objects. While early binding method, functions and properties which are detected and checked during compile time and perform other optimization before an application executes. The biggest advantage of using early binding is for performance and ease of development.

Example:

```
#include <iostream>
using namespace std;
class Base
{
  public:
      void show() {cout << "In Base \n";}
};
class Derived : public Base {
      void show() { cout << "In Derived \n";
                  }
};
int main() {
      Base * bp = new Derived;
      bp → show();
      return 0;
}
```

Output:

In Base.

Late Binding (Dynamic Binding):-

By Contrast, in late Binding functions, methods, variables and properties are detected and properties are detected and checked only at the run time. It implies that the compiler does not know what kind of object or actual type of an object or which methods or properties an object contains until run time. The biggest advantages of late binding is that the objects of this type can hold reference to any object.

Example:

```
#Include <iostream>
using namespace std;
Class Base
{ public:
        virtual void show() {cout<<"In Base
                                    /n";}
  };
  Class derived
   {
     public:
        virtual void show() {cout<<"In Base \n";}

    };
    Class derived
    {
      Public:
        void show() {cout" <<" In derived \n";}
    };
   int main() {
      Base *bp = new Derived;
      bp → show ();
      return 0;
   }
```

output:
In Derived

Unit - 3

Answer 3(a)

The Symbolic Expression (S-expression). The syntactic elements of the Lisp programming language are symbolic expressions, also known as S-expressions. .Both programs and data are represented as s-expressions: an S-expression may be either an atom or a list.

A symbolic expression as S-expression, S-exp, or S-exp is a way to represent a nested list of data in lisp. for example (*5(+ 73))

Answer 3(c)

~~Imp~~ Imperative Programming:

It is a ~~pro~~ paradigm of computer programming where the program of computer programming where the program describes steps that change the state of the computer. Unlike declarative programming, which describes "what" a program should accomplish, it. Programs written this way often compile to binary executables that run more efficiently since all CPU instructions are themselves imperative statements.

To make programs simpler for a human to read and write, imperative statements can be grouped into sections known as code blocks.

In the 1950s, the idea of grouping a program's code into blocks was first implemented in the ALGOL programming language. They were originally called "compound statements" but today these blocks of code are known as procedures.

Functions programming is a programming paradigm in which we try to bind everything in pure mathematical functions style. It is a declarative type of programming style. Its main focus is on "what to solve". It uses expressions instead of statements. An expression is evaluated to produce a value whereas a statement is executed to assign variables. Programming languages that support functional programming: Haskell, Javascript, Python, Scala, etc.

~~Unit 4~~

Answer 4(D)

Datatypes in Prolog:

There are nine Types of Datatypes in PROLOG:-

1.) Char: Character is enclosed between a pair of ~~sig~~ single quotes.

2.) Integer: A whole number in the ~~region~~ range of -32768 to 32767.

3.) Real: Positive or negative special character followed by digits.

4.) String: Any set of characters enclosed within a pair of double-quotes. strings can include up to 255 characters.

5.) Symbol: A sequence of letters (A to z or a to z) digits (0 to 9) and the special character underscore(_)

6.) Variables: A variable is a symbol that can be assigned different values at different stages of the execution of the program.

7.) Reserved words: PROLOG has some reserved words and it must not be used in place of user-defined names. Ex - and, asserta.

8.) Arthematic Operators: +, -, * and / are basic arithmetic operators in PROLOG.

9.) Relational Operations: PROLOG uses <, <=, =, >, >=, < > relational operators. In PROLOG, a relational operator can be goal or a subgoal. The relational operator (=) looks same as assignment operator.

Unit - 4

Answer 4(a)

(i) **Abstraction**: It is the concept of object - oriented programming that "shows" only essential attributes and ~~hig~~ "hides" unnecessary information. The main purpose of abstraction is hiding the unnecessay details from the users. Abstraction is selecting data from a larger pool to show only relevant details of the object to the user. It helps in reducing programming complexity and efforts. It is one of the most important concepts of OOPs.

(ii) Polymorphism: It is one of the core-concepts of OOP → ~~of~~ object oriented programming and ~~dec~~ describes situations in which something occurs in several different forms. In computer science, it describes the concepts that you can access objects of different types through the same interface. Each type can provide its own independent implementation of this interface.

Answer (4)(b)

Static Member Function:

1.) Member function becomes independent of object. Or shared by all of its objects.

2.) A static member function can be called without object. Means even no objects is created

3.) Static member functions are called using class name.
class_name :: function name().

★ A function is ~~made~~ made by using static keyword with function name.

★ It can be called using the object and the direct member access (.) operator.
But, its more typical to call a static member function by itself, using class name and scope resolution (::) operator.

Note : Restriction

≡ As a static member function is a kind of global for that particular class it can have only static data member. Or it can perform any operation, manipulation on only static data members.
>> A static member function cannot use non-static member.

Answer 4(D)

Operator new, new[]	Operator delete, delete[]
1.) new operator is used to allocate memory dynamically.	1.) delete operator is used to ~~del~~ deallocates or free the dynamic memory.
2.) new[] operator is used to allocate array of data type memory dynamically.	2.) delete[] operator is used to deallocate or free the dynamically. allocated memory of the array.
3.) new returns a pointer to the begning of new block. when memory allocate it.	3.) delete is freed, when memory allocated by new is no longer required.
4.) new[] returns a unique pointer to the beginning of new block when allocating an array using the new operator the first dimension can be zero.	4) delete[] deletes the array pointed value, returned by a pointer.

Syntax:

type pointer_variable = new type.

Synatax:

type pointer_variable = new type[].

Syntax:

delete pointer_variable.

Syntax:

delete[] pointer_variable.

Unit - 5

Answer 5(a)

Constructor: A constructor is a special type of member function of a ~~sp~~ class which initializes objects of a class. In C++, Constructor is automatically called when objects (instance of class) is created. It is special member function of the class because it ~~do~~ does not have any return type.

Destructor: A destructor is a member function that is invoked automatically when the object goes out of scope or is explicitly destroyed by a call to delete. A destructor has the same name as the class, preceded by a tilde(~). For example, the destructor for class string is declared.: ~string().

Answer 5(b)

Friend Function: If a function is friend function of a class, then the friend function is not the actual member of the class. But that friend function has rights to access to all private and protected members (variables and functions). Friend function like friend class, a friend function can be given a special grant to access private and protected members. A friend function can be:

(a) A member of another class.

(b) A global function.

~~Friend~~

Friend Function Syntax :

```
class class_name
{
  // member variables and function
  // Friend Function Declaration.
  friend return_type friend_function_name
  (arguments/objects);
}
// Friends Function Definition
return_type friend_function_name
(arguments / objects)
 {
  // friend function has privileges
  to access all private & protected
  members of the class.
}.
```

(ii) Abstract class: It is a class that is designed to be specifically used as a base class. An abstract class contains at least one pure virtual function. You declare a pure specifier (=0) in the declaration.

The following is an example of an abstract class:

```
class AB {
public:
    virtual void f() = 0;
}
```

Answer 5(c)

* **Virtual Function**: It is a member function which is declared within a base class and is re-defined (overridden) by a derived class. When refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.

* Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.

* They are mainly used to achieve Runtime polymorphism.
* Functions are declared with a virtual keyword in base class.
* The resolving of function call is done at runtime.

Rules for Virtual Functions:

1.) Virtual functions cannot be static.
2.) A virtual function can be a friend function of another class.
3.) ~~virtual~~ virtual function should be accessed using pointer or reference of base class type to achieve runtime polymorphism.
4.) The prototype of virtual function should be the same in the base as well as derived class.
5.) A class may have virtual destructor but it cannot have a virtual constructor

```
//cpp program to illustrate
//concept of virtual Functions.

#include<iostream>
using namespace std;
 Class base{
 public:
  virtual void print()
  {
   cout <<"print base class \n";
  }
  void show(),
    {
     cout <<"show base class \n";
    }
  };
   Class derived : public base{
   public:
   void print()
   {
     cout <<"print derived class \n";
    }
    void show()
    {
     cout <<"show derived class \n";
    }
};
```

```
int main()
{
 base * bptr;
 derived d;
 bptr = &d;
 // virtual function, bided at runtime
  bptr -> print();
 // Non-virtual function, binded at
 compile time
  bptr -> show();

  return 0;
 }
```