Principles of OOP :-

i) Abstraction
ii) Encapsulation
iii) Inheritance
iv) Polymorphism.

શુક્રવાર
ચૈત્ર સુદ ૭ શાકે ૨૨ ચૈત્ર
12

The Wrapping up of data & fun into a Single unit.

→ private, public, protected.

Scope resolution operat^{n} ::

→ fu^{n} definition

શનિવાર
ચૈત્ર સુદ ૮ શાકે ૨૩ ચૈત્ર
13
બીજો શનિવાર

Return-type classname::
fu^{n}-name(arg)
{
fu^{n} body;
}

રવિવાર
ચૈત્ર સુદ ૯ શાકે ૨૪ ચૈત્ર
14

શ્રી રામ નવમી, ડો. બાબાસાહેબ આંબેડકરનો જન્મદિન

એપ્રિલ ૨૦૧૯

15 સોમવાર
ચૈત્ર સુદ ૧૦ શાકે ૨૫ ચૈત્ર

milestones :-
discipline :- in mind 1st.

---

object as an arg in fun :-
Returning objects from a fun :
member fun calling other member fun.
memory allocat^n of objects.

16 મંગળવાર
ચૈત્ર સુદ ૧૨ શાકે ૨૬ ચૈત્ર

man sleep
man 3 sleep → header file shown

→ friend fun :-
1) friend class.
2) fun I in class A which is friend of class B.

18/2/22 Constructor :-
- always public

17 બુધવાર
ચૈત્ર સુદ ૧૩ શાકે ૨૭ ચૈત્ર
મહાવીર જયંતી

- shallow copy and deep copy

21/2/22.

Inline functions :-

23

inline return-type fu^n. name (arg)
{
fu^n code.
}

22/2/22

operator overloading

$C_3 = C_1$.operator+(C_2) fu^n notation

$C_3 = C_1 + C_2$ operator notation

24

overloading

Unary operator.

23/2/22.

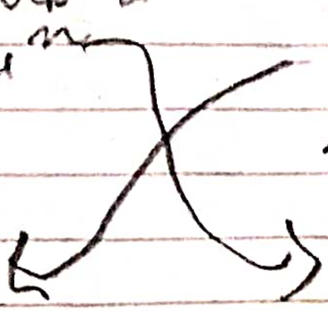
Task 1. Task :- 2

H.W. C_1 C_2 main :- class1 a_1, a_2

member m class2 b_1, b_2

fu^n f. $b_2 = a_1 + b_1$

f_1 f_2. b_2



25/4/22

operator

- We can not overload the member function of one class to Friend of another class.

શનિવાર
ચૈત્ર વદ ૫ શાકે ૫ વૈશાખ
25

- new: memory allocation.
- delete - Free up the memory.

26/4/22 H.W.

- overload following in new[], delete().
- class to basic type conversion.
  int, char.
- class to class type conversion.

રવિવાર
ચૈત્ર વદ ૬ શાકે ૬ વૈશાખ
26

શનિવાર
ચૈત્ર વદ ૮ શાકે ૭ વૈશાખ
ચોથો શનિવાર
27

રવિવાર
ચૈત્ર વદ ૯ શાકે ૮ વૈશાખ
28

c++ Notes:

Introduction to c++ and object oriented programming concepts. Object-Oriented Programming System (OOPs) is a programming concept
that works on the principles of abstraction, encapsulation, inheritance, and polymorphism. It allows users to create objects they want and create methods to
handle those objects. The basic concept of OOPs is to create objects, re-use them throughout the program, and manipulate these objects to get results.

Installing g++ and using g++ to compile c++ programs.

1)Class
The class is one of the Basic concepts of OOPs which is a group of similar entities. It is only a logical component and not the physical entity. Lets understand this one of the OOPs Concepts with example, if you had a class called “Expensive Cars” it could have objects like Mercedes, BMW, Toyota, etc. Its properties(data) can be price or speed of these cars. While the methods may be performed with these cars are driving, reverse, braking etc.

2) Object
An object can be defined as an instance of a class, and there can be multiple instances of a class in a program. An Object is one of the Java OOPs concepts which contains both the data and the function, which operates on the data. For example – chair, bike, marker, pen, table, car, etc.

3) Inheritance
Inheritance is one of the Basic Concepts of OOPs in which one object acquires the properties and behaviors of the parent object. It’s creating a parent-child relationship between two classes. It offers robust and natural mechanism for organizing and structure of any software.

4) Polymorphism
Polymorphism refers to one of the OOPs concepts in Java which is the ability of a variable, object or function to take on multiple forms. For example, in English, the verb run has a different meaning if you use it with a laptop, a foot race, and business. Here, we understand the meaning of run based on the other words used along with it. The same also applied to Polymorphism.

5) Abstraction
Abstraction is one of the OOP Concepts in Java which is an act of representing essential features without including background details. It is a technique of creating a new data type that is suited for a specific application. Lets understand this one of the OOPs Concepts with example, while driving a car, you do not have to be concerned with its internal working. Here you just need to concern about parts like steering wheel, Gears, accelerator, etc.

6) Encapsulation
Encapsulation is one of the best Java OOPs concepts of wrapping the data and code. In this OOPs concept, the variables of a class are always hidden from other classes. It can only be accessed using the methods of their current class. For example – in school, a student cannot exist without a class.

Dynamic Binding: In dynamic binding, the code to be executed in response to function call is decided at runtime. C++ has virtual functions to support this.

Message Passing: Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function and the information to be sent.

Access specifiers
All the variables and member function in a class will have one of following three properties: Public, Private or Protected.

Private Members
Private members of the class can be accessed within the class and from member functions of the class.
They cannot be accessed outside the class or from other programs, not even from inherited class.
If you try to access private data from outside of the class, compiler throws error.
This feature in OOP is known as Data hiding / Encapsulation.

Public Members
The public keyword makes data and functions public.
Public members of the class are accessible by any program from anywhere.
Passing Objects as Function Arguments
We can pass an object in the function argument just like any other data type We need to define the operations inside the function carefully
In Calculator prMemory allocation of objects
The member functions are created and placed in the memory space only once at the time they are defined as part of a class specification.
No separate space is allocated for member functions when the objects are created.
Only space for member variable is allocated separately for each object because, the member variables will hold different data values for different objects.
program we can modify the add() to take two objects as arguments

Friend Function
In C++ a Friend Function that is a "friend" of a given class is allowed access to private and protected data in that class.
It is declared using friend keyword
Friend function can be declared either in public or private part of the class. • It is not a member of the class so it cannot be called using the object. Usually, it has the objects as arguments.
Member functions of one class can be made friend function of another class using friend keyword in definition.

Static Data members
A static data member is useful, when all objects of the same class must share a common information.
Just write static keyword prefix to regular variable
It is initialized to zero when first object of class created Only one copy is created for each object
Its life time is entire program

Static member functions can access only static members of the class.
Static member functions can be invoked using class name, not object.

They cannot be virtual.
They cannot be declared as constant or volatile.
A static member function does not have this pointer.
They cannot be virtual.
They cannot be declared as constant or volatile.
A static member function does not have this pointer.

A constructor is a block of code which is,
similar to member function
has same name as class name
called automatically when object of class created
A constructor is used to initialize the objects of class as soon as the object is created.
Constructor should be declared in public section because private constructor cannot be invoked outside the class so they are useless.
Constructors do not have return types and they cannot return values, not even void.
Constructors cannot be inherited, even though a derived class can call the base class constructor.

Default Constructor
Default constructor is the one which invokes by default when object of the class is created.
It is generally used to initialize the default value of the data members.
It is also called no argument constructor.

Parameterized Constructor
Constructors that can take arguments are called parameterized constructors.
Sometimes it is necessary to initialize the various data elements of different objects with different values when they are created.
We can achieve this objective by passing arguments to the constructor function when the objects are created.

Copy Constructor
A copy constructor is used to declare and initialize an object from another object using an object as argument.
Constructor which accepts a reference to its own class as a parameter is called copy constructor.

Destructor
Destructor is used to destroy the objects that have been created by a constructor.
The syntax for destructor is same as that for the constructor,
the class name is used for the name of destructor,
with a tilde (~) sign as prefix to it.
never takes any argument nor it returns any value nor it has return type.
is invoked automatically by the complier upon exit from the program.

Functions
(inline, overloading, default args)

Inline Functions
Every time a function is called it takes a lot of extra time to execute series of instructions such as
- Jumping to the function
- Saving registers
- Pushing arguments into stack
- Returning to the calling function

If a function body is small then overhead time is more than actual code execution time so it becomes more time consuming.
Preprocessor macros is a solution to the problem of small functions in C.
In C++, inline function is used to reduce the function call overhead.

Inline Functions (Cont…)

Syntax:

```
inline return-type function-name(parameters) {
// function code
}
```

- Add inline word before the function definition to convert simple function to inline function.

Example:

```
inline int Max(int x, int y)
{
if (x>y)
return x;
else
return y;
}
```

Program: Inline function

- Write a C++ program to create inline function that returns cube of given number (i.e n=3, cube=(n*n*n)=27).

```
#include <iostream>
using namespace std;
inline int cube(int s)
{
return s*s*s;
}
int main()
{
cout << "The cube of 3 is: " << cube(3); return 0;
}
```

Critical situations Inline Functions

- Some of the situations inline expansion may not work

1) If a loop, a switch or a goto exists in function body.
2) If function is not returning any value.
3) If function contains static variables.
4) If function is recursive.

Default Arguments

while invoking a function If the argument/s are not passed then, the default values are used.
We must add default arguments from right to left.
We cannot provide a default value to a particular argument in the middle of an argument list.
Default arguments are useful in situations where some arguments always have the same value.

Function overloading

C++ provides function overloading which allows to use multiple functions sharing the same name .
Function overloading is also known as Function Polymorphism in OOP. It is the practice of declaring the same function with different signatures.

 However, the two functions with the same name must differ in at least one of the following,

 The number of arguments
 The data type of arguments
 The order of appearance of arguments

Arguments make the function unique

• Function overloading does not depend on return type.

Inheritance

Outline

• Concept of inheritance
• Types of inheritance
    1. Single
    2. Multiple
    3. Multilevel
    4. Hierarchical
    5. Hybrid
• Protected members
• Overriding
• Virtual base class

Concept of Inheritance class Doctor
Attributes:
Age, Height, Weight
Methods:
Talk()
Walk()
Eat()
Diagnose()
class Footballer
Attributes:
Age, Height, Weight Methods:
Talk()
Walk()
Eat()
Playfootball()
class Businessman
Attributes:
Age, Height, Weight Methods:
Talk()
Walk()
Eat()
Runbusiness()

 All of the classes have common attributes (Age, Height, Weight) and methods (Walk, Talk, Eat).
 However, they have some special skills like Diagnose, Playfootball and Runbusiness.
 In each of the classes, you would be copying the same code for Walk, Talk and Eat for each character.

Concept of Inheritance(Cont…)
class Person class Person is called Base

class Doctor
Methods:
Diagnose()
Attributes:
Age, Height, Weight Methods:
Talk() , Walk(), Eat()
class Footballer

Methods:
Playfootball()
These classes are called Derived class
class
class Businessman
Methods:
Runbusiness()

Inheritance

Inheritance is the process, by which class can acquire(reuse) the properties and methods of another class.
The mechanism of deriving a new class from an old class is called inheritance.
The new class is called derived class and old class is called base class.
The derived class may have all the features of the base class and the programmer can add new features to the derived class.

Types of Inheritance
1. Single Inheritance
2. Multilevel Inheritance
3. Multiple Inheritance
4. Hierarchical Inheritance
5. Hybrid Inheritance (also known as Virtual Inheritance)

Single Inheritance A
If a class is derived from a single class then it is called single inheritance.
Class B is derived from class A

Multilevel Inheritance
A B
C
Any class is derived from a class which is derived from another class then it is called multilevel inheritance.
Here, class C is derived from class B and class B is derived from class A, so it is called multilevel inheritance.

Multiple Inheritance
A B
C
Here, class C is derived from two classes, class A and class B.

Hierarchical Inheritance
A
B C D
If one or more classes are derived from one class then it is called hierarchical inheritance.
Here, class B, class C and class D are derived from class A.

Hybrid Inheritance
A
B C D
A and class D is derived from class B and class C.
Class A, class B and class C is example of Hierarchical Inheritance and class B, class C and class D is example of Multiple Inheritance so this hybrid inheritance is combination of Hierarchical and Multiple Inheritance.
It is a combination of any other inheritance types. That is either multiple or multilevel or hierarchical or any other combination.
Here, class B and class C are derived from class.

//girls

I/O and file handling

Copy content of one file to another:
Output file handling

Create new file and write on it

Open existing file and overwrite it

Open existing file and append new data at the end

Open existing file and write at any location

File handling in c++ library

• C++ standard library called fstream, defines the following classes to support file handling
• ofstream class: Provides methods for writing data into file. open(), put(), write(), seekp(), tellp(), close(), etc.

• ifstream class: Provides methods for reading data from file.

open(), get(), read(), seekg(), tellg(), close(), etc.
• fstream class: Provides methods for both writing and reading data from file. This includes all the methods of ifstream and ofstream class.

Using read() and write() objects

Syntax:

```
fstream fin;
fin.read((char *)&obj, sizeof(obj));
```

Using read() and write() objects

Syntax:

```
fstream fout;
fout.write((char *)&obj, sizeof(obj));
```

Types of file access:
1) Sequential access:
2) Random access:
File pointers

• Each file object has two integer values associated with it:

• get pointer
• put pointer

Random file access

• get and put stream pointers:
• tellg() – tellp()
• seekg() – seekp()

Functions to move the file pointer

• seekg() and tellg() are used to modify the 'get pointer'

• seekp() and tellp() are used to modify the 'put pointer'

Day 18
Templates in c++

Function templates
Class templates
Variable templates

Function templates:
Templates use less codes and easier to manage

How to define template:

```
template < class T >
T fun_name (T argument)
{
}
```

Function template

```
template <class T>
template <typename T>
```

Class template
A class template can represent various similar
classes operating on different

Template function overloading

If there are more than one function of the same name,

which differ only by number and/or types of
parameters, it is called function overloading

If at least one of them is a template function, then it is
called template function overloading

Class templates and static variables

The rule for the class template is similar to that of
function templates

Template specialization: compile time

When we write any template based function or class,
compiler creates a copy of that function/class
whenever compilers sees that being used for a new
data type

**Graphs
Graphs

A graph is a non-linear data structure. A graph can be defined as a collection of Nodes which are also called “vertices” and “edges” that connect two or more vertices

A graph can also be seen as a cyclic tree where vertices do not have a parent-child relationship but maintain a complex relationship among them.
Graphs: types

Direction
Directed
Un-directed
Graphs: types

Weight
Weighted
Un-weighted
Graphs: terminology

Vertex: Each node of the graph is called a vertex.

Edge: The link or path between two vertices is called an edge. It connects
two or more vertices.

- Adjacent node: In a graph, if two nodes are connected by an edge then they are called adjacent nodes or neighbors.

- Degree of the node: The number of edges that are connected to a particular node is called the degree of the node.

- Path: The sequence of nodes that we need to follow when we have to travel from one vertex to another in a graph is called the path.

- Closed path: If the initial node is the same as a terminal node, then that
path is termed as the closed path.

Graphs: representation

Sequential Representation: In the sequential representation of graphs, we use the adjacency matrix. An adjacency matrix is a matrix of size nxn where n is the number of vertices in the graph

The rows and columns of the adjacency matrix represent the vertices in a graph.
The matrix element is set to 1 when there is an edge present between the vertices. If the edge is not present then the element is set to 0

Graphs: representation

Linked Representation: We use the adjacency list for the linked representation of the graph.

The adjacency list representation maintains each node of the graph and a link to the nodes that are adjacent to this node.

When we traverse all the adjacent nodes, we set the next pointer to null at the end of the list.

Graphs: representation

Un-directed un-weighted graph

**Design Patterns
What is design patterns

• Software design patterns are abstractions that help structure system designs
• A design pattern is neither a static solution, nor is it an algorithm.
• A pattern is a way to describe and address by name a repeatable solution
or approach to a common design problem, that is, a common way to solve a generic problem
• Patterns are commonly found in objected-oriented programming languages like C++ or Java.
• They can be seen as a template for how to solve a problem that occurs in
many different situations or applications.
• It is not code reuse, as it usually does not specify code, but code can be
easily created from a design pattern

Creational

• Builder
• Factory method
• Abstract Factory
• Prototype
• Singleton

Structural

- Adapter
- Bridge
- Composite
- Decorator
- Façade
- Flyweight
- Proxy
- ....

Behavioural
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- Visitor
- Template method

Depending on the design problem they address, design patterns can be classified in different categories, of which the main categories are:
**Shared library
- You can also create a shared library that can be installed on a machine
- A shared library will have .so extension
- It can be installed using sudo make install command
- Then you can add the installed folder to the path variable

or

- Closed path: If the initial node is the same as a terminal node, then wide

Problem/requirement
To use a design pattern, we need to go through a mini analysis design that may be coded to test out the solution. This section states the requirements of the problem we want to solve. This is usually a common problem that will occur in more than one application.
Forces
This section states the technological boundaries, that helps and guides the creation of the solution.
Solution
This section describes how to write the code to solve the above problem. This is the design part of the design pattern. It may contain class diagrams, sequence diagrams, and or whatever is needed to describe how to code the solution.

Creational Patterns : Builder

It is used to separate the construction of a complex object from its representation so that the same construction process can create different objects representations.

Problem
We want to construct a complex object, however we do not want to have a complex constructor member or one that would need many arguments.
Solution

Define an intermediate object whose member functions define the desired object
part by part before the object is available to the client. Builder Pattern lets us
defer the construction of the object until all the options for creation have been
specified.

Creational Patterns : Factory
• A utility class that creates an instance of several families of classes
• It is useful in a situation that requires the creation of many different types
of objects, all derived from a common base type.

Problem
We want to decide at run time what object is to be created based on some configuration or application parameter. When we write the code, we do not know
what class should be instantiated.
Solution
Define an interface for creating an object, but let subclasses decide which class
to instantiate. Factory Method lets a class defer instantiation to subclasses.

Creational Patterns : Prototype
A prototype pattern is used in software development when the type of objects
to create is determined by a prototypical instance, which is cloned to produce
new objects.

Creational Patterns : Singleton
It ensures that a class has only one instance and provides a global point of
access to that instance

• Define a private static attribute in the "single instance" class.
• Define a public static accessor function in the class.
• Do "lazy initialization" (creation on first use) in the accessor function.
• Define all constructors to be protected or private.
• Clients may only use the accessor function to manipulate the Singleton.
Ex: You want to create a database connection, and want to restrict it to the
limited number of connections. You can create singleton class object for the
connection, and it will not allow more members to connect with the databse

Structural Patterns : Adapter
Convert the interface of a class into another interface that clients expect.
Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

Power adapters: US, INDIA, EUROPE all has different adapters to connect the charger from the socket

In a java based web application you may want to connect to some services that accept json or xml requests, the adapters will be a handy tools to design
such applications
**Shared library

- A statically-linked library is created at compile time to contain all of the
code relating the library
- Adjacent node: In a graph, if two nodes are connected by an edge then other libraries.
- This results in a library that is typically larger in size than the equivalent
shared library, but because all of the dependencies are determined at compile time, there are fewer run-time loading costs and the library may be more platform independent.
- Adjacent node: In a graph, if two nodes are connected by an edge then use a
shared library

Static library

Some reading material describing the steps to create
shared/static library will be shared in the classroom

**Project with directories

-As your project grows, it is likely that you will organize it into sub-directories.

-It is usual practice to place an individual Makefile in each sub-directory.
-These Makefiles are then individually invoked by the Makefile in the parent directory.
-CMake can be very useful in this situation. In this example, a project with
a typical directory structure is used. The tree utility program displays below the structure of the example project


```
$ tree
.
|-- CMakeLists.txt
|-- build
|-- include
| \-- Student.h
\-- src
|-- Student.cpp
\-- mainapp.cpp
3 directories, 4 files
```

The Multi-directory Project CMakeLists.txt file

```
cmake_minimum_required(VERSION 2.8.9)
project(directory_test)
```

```
#Bring the headers, such as Student.h into the project
include_directories(include)

#Can manually add the sources using the set command as
follows:
#set(SOURCES src/mainapp.cpp src/Student.cpp)

#However, the file(GLOB...) allows for wildcard
additions:
file(GLOB SOURCES "src/*.cpp")

add_executable(testStudent ${SOURCES})
```

The important changes in here are as follows:

- The include_directories() function is used to bring the header files into the build environment.
- You can also create a shared library that can be installed on a machine that contains the name values of all of the source files (.cpp) in the project. However, because each source file must be added manually the next line is used in its place, and this line is commented out.
- The file() command is used to add the source files to the project. GLOB (or GLOB_RECURSE) is used to create a list of all of the files that meet the globbing expression (i.e., "src/*.cpp") and add them to a variable SOURCES.
- You can also create a shared library that can be installed on a machine explicit reference to each source file, in order to build the testStudent executable program.

Now we will create a build directory and call make from it

```
$ mkdir build
$ cd build
$cmake . .
```

Here the . . Is to tell that the cmakefile is located in the previous directory

Check the contents of the build directory now

```
$ls
```

It must have a Makefile, and some other support files
You can now run make command

```
$make
```

This will create a testStudent executable inside build directory

```
$ls
```

Now go back to the student directory (one level back)

```
$cd ..
```

And run

```
$ tree
|-- CMakeLists.txt
```

```
|-- build
| |-- CMakeCache.txt
| |-- CMakeFiles
| | |-- CMakeCCompiler.cmake
| | `-- testStudent.dir
| | |-- CXX.includecache
| | |-- progress.make
| | `-- src
| | |-- Student.cpp.o
| | `-- mainapp.cpp.o
| |-- Makefile
| |-- cmake_install.cmake
| `-- testStudent
|-- include
| `-- Student.h
`-- src
|-- Student.cpp

Output here is truncated

To clean this project you can remove all the files in the build directory
$ rm -r build/*

This will restore the file system structure to same as it was before
executing cmake command.

*cmake

cmake

• The make utility and Makefiles provide a build system that can be used
to
manage the compilation and re-compilation of programs that are written
in any programming language

• CMake is a cross-platform Makefile generator! Simply put, CMake
automatically generates the Makefiles for your project.


Hello world

#include<iostream>

int main(int argc, char *argv[]){
std::cout << "Hello World!" << std::endl;
return 0;
}


CMakeLists.txt


¯ The first line sets the minimum version of CMake for this project,
which is major
version 2, minor version 8, and patch version 9 in this example.
¯ Closed path: If the initial node is the same as a terminal node, then
this
example is determined just below.
¯ This results in a library that is typically larger in size than the
¯ Closed path: If the initial node is the same as a terminal node, then
```

executable is to be built using the helloworld.cpp source file.
¯ The first argument to the add_executable() function is the name of the executable
to be built, and the second argument is the source file from which to build the
executable

```
cmake_minimum_required(VERSION 2.8.9)
project (hello)
add_executable(hello helloworld.cpp)
```

Install cmake

Object oriented programming (C++), Pranav Verma

```
$sudo apt-get install cmake
```

```
$cmake -version
```

run cmake

```
$cmake .
```

execute the cmake command and pass it the directory that contains the source code and the CMakeLists.txt file — in this case ".” refers to the current directory:

Template function overloading

If there are more than one function of the same name, which differ only by number and/or types of parameters, it is called function overloading

If at least one of them is a template function, then it is called template function overloading

Template function overloading

If there are more than one function of the same name, which differ only by number and/or types of parameters, it is called function overloading

If at least one of them is a template function, then it is called template function overloading variables

Template function overloading

If there are more than one function of the same name, which differ only by number and/or types of parameters, it is called function overloading

If at least one of them is a template function, then it is called template function overloading