KARNATAK LAW SOCIETY’S

GOGTE INSTITUTE OF TECHNOLOGY

UDYAMBAG, BELAGAVI-590008

(An Autonomous Institution under Visvesvaraya Technological University, Belagavi)

**(APPROVED BY AICTE, NEW DELHI)**

Department of Computer Science Engineering



**OOP using C++ Journal**

2nd Semester B.E.

Submitted by

**PARITOSH KUMAR - 2GI20CS082**

Under the guidance of

**Prof. Manjula Ramannavar**

**2020-2021**

**Karnataka Law Society’s**

**GOGTE INSTITUTE OF TECHNOLOGY**

Udyambag Belagavi -590008,Karnataka, India.

**Department of Computer Science and Engineering**



**Certificate**

This is to certify that the journal term workscarried out by **Paritosh kumar** bearing **usn: 2gi20cs082** is submitted in partial fulfilment of the requirements for 2nd semester b.e.in **oop using c++, computer science and engineering,** visvesvaraya technological university, belagavi. It is certified that all corrections/suggestions indicated have been incorporated in the journal. The journal has beenapproved as it satisfies the academic requirements in respect of research work prescribed forthe said degree.

Date: Signature of guide

Place: Belagavi **Prof Manjula R**

Professor, dept. Of cse,

Kls gogte institute technology,belagavi

Name of the examiners signature of the examiners

1.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 1.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**INDEX**

|  |  |
| --- | --- |
| 1 | Program on classes and objects. |
| 2 | Program illustrating use of reference type in C++. |
| 3 | Program on function overloading. |
| 4 | Program on dynamic memory management in C++. |
| 5 | Program on array of objects. |
| 6 | Program on constructors and destructors. |
| 7 | Program on operator overloading. |
| 8 | Program on overloading i/o operators. |
| 9 | Program on inheritance. |
| 10 | Program on virtual function and pure virtual functions. |
| 11 | Program on dynamic polymorphism. |
| 12 | Program on file streams. |

**Term Work No.:1**  **Date:9-09-2021**

**Problem Statement:**

Create a class called Employee with name, ID, department and basic salary as data members.

Include member functions to:

a. Set values for data members,

b. Compute gross salary where DA=70% of basic, HRA=20% of basic and deductions=5% of

basic and

c. Display details of an employee.

**Objectives of the Experiment:**

1. To understand and apply the concept of classes
2. To implement modular programming

**Program:**

#include <iostream>

using namespace std;

class Employee {

private:

string name, dept;

int basic, grossSal;

public:

void initialize(string name, string dept, int basic) {

this->name = name;

this->dept = dept;

this->basic = basic;

}

void computeSalary() {

grossSal = basic + basic\*0.7 + basic\*0.2 - basic\*0.05;

}

void display() {

cout<< "Name:" << name <<endl;

cout<< "Department:" << dept <<endl;

cout<< "Basic Salary:" << basic <<endl;

cout<< "Gross Salary:" <<grossSal<<endl;

}

};

int main() {

Employee e1;

e1.initialize("Aniket", "CSE", 60000);

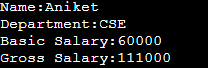
e1.computeSalary();

e1.display();

return 0;

}

**Output:**



**Outcomes of the Experiment:**

1. The concept of classes was demonstrated
2. The concept of modular programming was implemented

**Conclusions:**

From the given problem statement, we could understand and demonstrate the concept of classes in C++.

**Title of the Experiment**

**Program on classes and objects**

**Term Work No:2 Date:**

**Problem Statement:**

**2.1) Write a C++ program to swap two variables using reference variables.**

**Program Source Code:**

void swap(int &a, int &b) {

int temp;

temp = a;

a = b;

b = temp;

}

int main() {

int m = 10, n= 20;

cout<< "Before exchange m=" << m << " and n=" << n <<endl;

swap(m, n);

cout<< "After exchange m=" << m << " and n=" << n <<endl;

return 0;

}

**OUTPUT:**

**Before exchange m=10 and n=20**

**After exchange m=20 and n=10**

**Program ended with exit code: 0**

**Title of the Experiment**

**Program on function overloading**

**Term Work No.:3 Date:9-09-2021**

**Problem Statement:**

**Write a C++ program to calculate the area of circle (given the radius), rectangle (given the two**

**sides) and triangle (given the three sides) using function overloading.**

**Objectives of the Experiment:**

1. To understand and apply the concept of fuction overloading
2. To implement modular programming

**Program:**

#include <iostream>

#include <math.h>

using namespace std;

float computeArea(int rd) {

return 3.142 \* rd \* rd;

}

float computeArea(int len, int wid) {

return len \* wid;

}

float computeArea(int s1, int s2, int s3) {

float s = (s1+s2+s3)/2;

return sqrt(s\*(s-s1)\*(s-s2)\*(s-s3));

}

int main() {

cout<< "Area of circle with radius 7 is " <<computeArea(7) <<endl;

cout<< "Area of rectangle with sides 7 and 10 is " <<computeArea(7,10) <<endl;

cout<< "Area of triangle with sides 3,4 and 5 is " <<computeArea(3,4,5) <<endl;

return 0;

}

Output:



**Title of the Experiment**

**Program on dynamic memory management**

**Term Work No.:4**  **Date:9-09-2021**

**Problem Statement:**

Create an array using dynamic memory allocation. Write functions to perform the following:

a. Find the minimum element in the array

b. Find the maximum element in the array

c. Find the mean of the elements in the array

**Objectives of the Experiment:**

1. To understand and apply the concept of dynamic memory management
2. To implement modular programming

**Program:**

#include <iostream>

using namespace std;

void readArray(int a[], int n) {

cout<< "Enter " << n << " elements:";

for(int i=0; i<n; i++)

cin>> a[i];

}

void findMin(int a[], int n) {

int minimum=a[0];

for(int i=1; i<n; i++)

if(a[i]<minimum)

minimum = a[i];

cout<< "Minimum element is " << minimum <<endl;

}

void findMax(int a[], int n) {

int maximum=a[0];

for(int i=1; i<n; i++)

if(a[i]>maximum)

maximum = a[i];

cout<< "Maximum element is " << maximum <<endl;

}

void findMean(int a[], int n) {

int sum = 0;

for(int i=0; i<n; i++)

sum += a[i];

cout<< "Mean of the elements is " << sum/n <<endl;

}

int main() {

int n;

cout<< "How many elements?"<<endl;

cin>> n;

int \*a = new int[n];

readArray(a, n);

findMin(a, n);

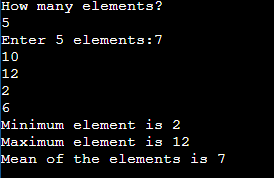
findMax(a, n);

findMean(a, n);

return 0;

}

**Output:**



**Outcomes of the Experiment:**

1. The concept of dynamic memory management was demonstrated
2. The concept of modular programming was implemented

**Conclusions:**

From the given problem statement, we could understand and demonstrate the concept of dynamic memory management in C++.

**Term Work No.:5**  **Date:9-09-2021** **Program on array of objects**

5.1) Create a Book class containing data members viz., Book number, Title, Author and

price. Write the main function that does the following:

            a. Creates an array of book objects and reads the book information using member

function.

            b. Given the book number, searches and prints using friend function, the book details

if found, an error message otherwise.

**Source code:**

class Book {private: int bookNo,price;

string title, author;

public: void readBookInfo() {

cout<< "Enter book number:"; cin>>bookNo;

fflush(stdin); cout<< "Enter title:"; getline(cin, title);fflush(stdin); cout<< "Enter author:"; getline(cin, author);

fflush(stdin); cout<< "Enter price:";cin>> price;}

friend void searchBook(Book b[], int n, int bNo);}};

void searchBook(Book b[], int n, int bNo) {

int i; for(i=0; i<n; i++)

if(b[i].bookNo == bNo) break;

if(i<n) { cout<< "Book Found. Details are..." <<endl;

cout<< "Title:" << b[i].title<<endl; cout<< "Author:" << b[i].author<<endl; cout<< "Price:" << b[i].price<<endl; }

else

cout<< "Such book number does not exist!" <<endl;}

int main() {

int n;

cout<< "How many books?";

cin>> n;

Book \*b = new Book[n];

cout<< "Enter details of " << n << " books..." <<endl;

for(int i=0; i<n; i++)

b[i].readBookInfo();

int bNo;

cout<< "Enter book number to search:";

cin>>bNo;

searchBook(b, n, bNo);

return 0;

}

**OUTPUT:**

**How many books?**3

**Enter details of 3 books...**

**Enter book number:**101

**Enter title:**egr mech

**Enter author:**gupta

**Enter price:**350

**Enter book number:**102

**Enter title:**rc bhargava

**Enter author:**rc bh

**Enter price:**200

**Enter book number:**104

**Enter title:**oop in c++

**Enter author:**e balaguruswamy

**Enter price:**380

**Enter book number to search:**104

**Book Found. Details are...**

**Title:oop in c++**

**Author:e balaguruswamy**

**Price:380**

**Program on constructors and destructors**

**Termwork 6:**

Create a class called intArray with

Data members:

1. pointer to an integer array and

2. integer to hold the array length

Member functions:

1. A zero-arg constructor

2. A parameterized constructor with an array and its length as parameters

3. A copy constructor

4. Display array elements

5. Destructor

Write the corresponding main()

Source code:

class IntArray {private: int len, \*arr;

public: IntArray() { len = 10;

arr = new int[len]; for(int i=0; i<len; i++)

arr[i]=0;} IntArray(int a[], int n) {

len = n; arr = new int[len];

for(int i=0; i<len; i++)

arr[i] = a[i]; }

IntArray(const IntArray&a) {

len = a.len; arr = new int[len];

for(int i=0; i<len; i++) arr[i] = a.arr[i];}

void printArray() { for(int i=0; i<len; i++)

cout<<setw(3) <<arr[i]; }};

int main() { cout<< "Creating a default IntArray object..." <<endl; IntArray ob1;

cout<< "Default IntArray object contents are:" <<endl;

ob1.printArray();

cout<<endl<< "Creating parameterized IntArray object..." <<endl;

int a[] = {1,2,3,4,5};

IntArray ob2(a,5);

cout<< "Parameterized IntArray object contents are:" <<endl;

ob2.printArray();

cout<<endl<< "Creating IntArray object using copy constructor..." <<endl;

IntArray ob3(ob2);

cout<< "Contents of copied IntArray object are:" <<endl;

ob3.printArray();

return 0;

}

**Out[put:**

**Creating a default IntArray object...**

**Default IntArray object contents are:**

**0  0  0  0  0  0  0  0  0  0**

**Creating parameterized IntArray object...**

**Parameterized IntArray object contents are:**

**1  2  3  4  5**

**Creating IntArray object using copy  constructor...**

**Contents of copied IntArray object are:**

**1  2  3  4  5**

**Title of the Experiment**

**Program on operator overloading**

**Term Work No.:7**  **Date:9-09-2021**

**Problem Statement:**

Create a class called Complex with real and imaginary parts as data members.

Member functions:

a. A zero-argument constructor

b. A parameterized constructor

c. Overloaded + operator to add two complex numbers and return the sum

d. Display complex number

Write the corresponding main()

**Objectives of the Experiment:**

1. To understand and apply the concept of operator overloading
2. To implement modular programming

**Program:**

#include <iostream>

using namespace std;

class Complex {

private:int real; int imaginary;

public:Complex() {real = imaginary = 0;}

Complex(int r, int i) {

real = r;imaginary = i;}

void print() {cout<< real << "+i" << imaginary <<endl;}

friend Complex operator+(Complex a, Complex);};

Complex operator+(Complex a, Complex b) {

Complex res;

res.real = a.real + b.real;

res.imaginary = a.imaginary + b.imaginary;return res;}

int main() {

Complex c1(7,9), c2(4,6);

cout<< "Complex c1 = "; c1.print();

cout<< "Complex c2 = ";

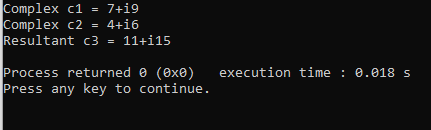
c2.print();Complex c3; c3 = c1 + c2;

cout<<"Resultant c3 = ";

c3.print();

return 0;}

**Output:**



**Outcomes of the Experiment:**

1. The concept of operator overloading was demonstrated
2. The concept of modular programming was implemented

**Conclusions:**

From the given problem statement, we could understand and demonstrate the concept of operator overloading in C++.

**Termwork:8**

**8) Program on overloading i/o operators**

8.1) Create a class called Distance with feet and inches as data members.

Member functions:

1. A zero-argument constructor

2. A parameterized constructor

Friend functions:

a. Overloaded extraction operator to read a distance

b. Overloaded insertion operator to display a distance (Ex - 10 feet 5 inches)

c. Compare two distances

Write the corresponding main()

**Source code:**

class Distance {

private:int feet;

int inches;

public:

Distance() { feet = 0;

inches = 0;}

Distance(int f, int i) {

feet = f;

inches = i; }

friend ostream&operator<<(ostream&output, const Distance &D) {

output<< "F : " <<D.feet<< " I : " <<D.inches;

return output;

}

friend istream&operator>>(istream&input, Distance &D) {

input>>D.feet>>D.inches;

return input;

}

friend void compare(Distance d1, Distance d2); };

void compare(Distance d1, Distance d2) {

int ob1toInch = d1.feet \* 12 + d1.inches;

int ob2toInch = d2.feet \* 12 + d2.inches;

if(ob1toInch < ob2toInch)

cout<< "Two distances are same" <<endl;

else if(ob1toInch < ob2toInch)

cout<< "First distance is less than the second" <<endl;

else

cout<< "First distance is more than the second" <<endl;}

int main() {

Distance d1(11, 10), d2(5, 11), d3;

cout<< "Enter the value of object : " <<endl;

cin>> d3;

cout<< "First Distance : " << d1 <<endl;

cout<< "Second Distance :" << d2 <<endl;

cout<< "Third Distance :" << d3 <<endl;

compare(d1, d2);

return 0;

}

**Output:**

**Enter the value of object :**

4

6

**First Distance : F : 11 I : 10**

**Second Distance :F : 5 I : 11**

**Third Distance :F : 4 I : 6**

**First distance is more than the second**

**Title of the Experiment**

**Program on inheritance**

**Term Work No.:9**  **Date:11-09-2021**

**Problem Statement:**

Create a class called Manager with attributes: name, ID and basic salary. Derive two classes HR

Manager and Sales Manager from Manager and compute gross salary as per the following:

HR Manager - DA = 70% of basic, HRA = 20% of basic, deductions = 5% of basic

Sales Manager - DA = 70% of basic, HRA = 10% of basic, TA = 5% of basic, deductions = 5% of

Basic

**Objectives of the Experiment:**

1. To understand and apply the concept of inheritance
2. To implement modular programming

**Program:**

#include <iostream>

using namespace std;

class Manager {

private: int empID; string name;

protected: float basic, grossSalary;

public: Manager(int empID, string name, float basic) {

this->empID = empID;

this->name = name;

this->basic = basic;}

void dispDetails() {

cout << "Manager ID: " << empID << endl;

cout << "Name: " << name << endl;

cout << "Gross Salary is Rs." << grossSalary << endl;}};

class SalesManager : public Manager {

public: SalesManager(int empID, string name, float basic) : Manager(empID, name, basic) {}

void computeSalary() {

float da = 0.7 \* basic;

float hra = 0.1 \* basic;

float ta = 0.05 \* basic;

float deductions = 0.05 \* basic;

grossSalary = basic + da + hra - deductions;

}

};

class HRManager : public Manager {

public:

HRManager(int empID, string name, float basic) : Manager(empID, name, basic) {}

void computeSalary() {

float da = 0.7 \* basic;

float hra = 0.2 \* basic;

float deductions = 0.05 \* basic;

grossSalary = basic + da + hra - deductions; }};

int main() { SalesManager s1(234, "Aniket", 200000);

HRManager h1(756, "Maniprasad", 150000);

s1.computeSalary();

s1.dispDetails();

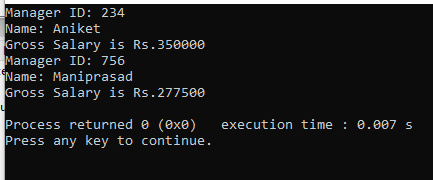
h1.computeSalary();

h1.dispDetails();

return 0;

}

**Output:**



**Title of the Experiment**

**Program on virtual function and pure virtual function**

**Term Work No.:10**  **Date:11-09-2021**

**Problem Statement:**

Create a base class called Student and derived classes UGStudent and PGStudent. Demonstrate

polymorphism using virtual functions to compute the total score as follows:

    UGStudent - average of best two scores

    PGStudent - addition of two scores

**Objectives of the Experiment:**

1. To understand and apply the concept of virtual function and pure virtual function
2. To implement modular programming

**Title of the Experiment**

**Program on virtual function and pure virtual function**

**Term Work No.:10**  **Date:11-09-2021**

**Problem Statement:**

Create a base class called Student and derived classes UGStudent and PGStudent. Demonstrate

polymorphism using virtual functions to compute the total score as follows:

    UGStudent - average of best two scores

    PGStudent - addition of two scores

**Objectives of the Experiment:**

1. To understand and apply the concept of virtual function and pure virtual function
2. To implement modular programming

**Program:**

#include <iostream>

#include <math.h>

using namespace std;

class Student {

protected:

int m1, m2, m3, avg;

public:

Student(int m1, int m2, int m3) {

this->m1 = m1;

this->m2 = m2;

this->m3 = m3;

}

Student(int m1, int m2) {

this->m1 = m1;

this->m2 = m2;

}

virtual void findTotal() { }

};

class UGStudent : public Student {

public:

UGStudent(int m1, int m2, int m3) : Student(m1, m2, m3) {}

void findTotal() {

int smallest = (m1<m2) ? (m1<m3 ? m1 : m3) : (m2<m3 ? m2 : m3);

avg = ceil((m1+m2+m3-smallest)/2.0);

cout << "Average score of UG Student is " << avg << endl;

}

};

class PGStudent : public Student {

public:

PGStudent(int m1, int m2) : Student(m1, m2) {}

void findTotal() {

avg = m1+m2;

cout << "Total score of PG Student is " << avg << endl;

}

};

int main() {

UGStudent ug(38,48,46);

PGStudent pg(41,43);

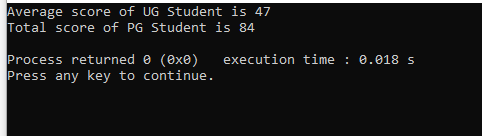
ug.findTotal();

pg.findTotal();

return 0;

}

**Output:**



**Outcomes of the Experiment:**

1. The concept of virtual function and pure virtual function was demonstrated.
2. The concept of modular programming was implemented

**Conclusions:**

From the given problem statement, we could understand and demonstrate the concept of virtual function and pure virtual function in C++.

**Termwowk 11:**

**Program on dynamic polymorphism**

11.1) Create a base class called List that has virtual functions: CreateList and DispList.

Create two classes IntList and CharList that inherit from List. Both classes override

CreateList and DispList. The IntList class has an additional member function to return the

sum of elements of the list of integers. The CharList has an additional member function to search for a character received as a parameter and returns the position of it’s first occurrence, if found in the list of characters; else returns

-1. Demonstrate dynamic polymorphism by

creating and upcasting objects of the two derived types in main.

**Source code:**

class List {

public: virtual void createList() = 0;

virtual void dispList() = 0;};

class IntList : public List {

private: int \*p; int n;

public: void createList() {

cout << "How many elements?"; cin >> n;

p = new int[n]; cout << "Enter " << n << " elements:" ;

for(int i=0; i<n; i++)

cin >> p[i];}

void dispList() {

cout << "The list has following elements:" << endl;

for(int i=0; i<n; i++)

cout << p[i] << " "; cout << endl;}

int findSum() { int sum = 0;

for(int i=0; i<n; i++) sum += p[i];

return sum;}};

class CharList : public List {

private: char \*p; int n;

public: void createList() {

cout << "How many characters?";

cin >> n; cin.ignore(80,'\n');

p = new char[n+1];

cout << "Enter " << n << " characters:" << endl;

int i;

char ch;

for(i=0; i<n; i++) {

cin.get(p[i]);

cin.ignore(80,'\n');

}

p[i] = '\0';

}

void dispList() {

cout << "The character list is " << p << endl;

}

int searchChar(char ch) {

for(int i=0; i<n; i++)

if(p[i] == ch)

return i;

return -1;

}

};

int main() {

IntList arr1;

List \*p = &arr1;

p->createList();

p->dispList();

cout << "Sum of integet list is " << arr1.findSum() << endl;

CharList arr2;

p = &arr2;

p->createList();

p->dispList();

char ch;

cout << "Enter character to search:";

cin >> ch;

int position = arr2.searchChar(ch);

if(position == -1)

cout << ch << " is not found in the list" << endl;

else

cout << ch << " is found at position " << position << endl;

return 0;

}

**OUTPUT:**

**How many elements?**3

**Enter 3 elements:**

4 7 34

**The list has following elements:**

**4 7 34**

**Sum of integet list is 45**

**How many characters?**3

**Enter 3 characters:**

r

g

k

**The character list is rgk**

**Enter character to search:**k

**k is found at position 2**

**TERMWORK 12:**

**Program on file streams**

Write a program that creates a file by reading and storing user input text. The program

further reads the file and creates another file that contains all the text of the input file

converted into uppercase.

***Source code***:

int main() {

string line;

char ch;

ofstream outFile("file1.txt");

cout << "Enter text. Press QUIT to terminate on a line:" << endl;

do {

getline(cin, line);

outFile << line << "\n";

}while(line!="QUIT");

outFile.close();

ifstream inFile("file1.txt");

outFile.open("file2.txt");

while(inFile.get(ch)) {

outFile.put(toupper(ch));

}

return 0;

}

**Output:**

**Enter text. Press QUIT to terminate on a line:**

Have a good day

Today is monday

Hello world

QUIT

**Program ended with exit code: 0**