

OOP Assignment

a) what is meant by polymorphism? Explain the types of polymorphism with an example of each.

Ans. Polymorphism simply means more than one form.

- That is, the same entity (function or operator) behaves differently in different scenarios.

→ Types of polymorphism:-

i) Compile Time polymorphism:-

- You invoke the overloaded functions by matching the number and type of arguments.

The information is present during compile-time.

1. Function overloading:-

- Function overloading occurs when we have many functions with similar names but different arguments.

The arguments may differ in terms of number or type.

e.g:

```
#include<iostream>
```

```
using namespace std;
```

```
int sum(int num1, int num2)
```

2

```
return num1 + num2;
```

3

```
double sum(double num1, double num2)
```

4

```
return num1 + num2;
```

5

```
int sum(int num1, int num2, int num3)
```

6

```
return num1 + num2 + num3;
```

ont main()

{

cout << "sum1 = " << sum(5,6) << endl;

cout << "sum2 = " << sum(5, 6, 5,7) << endl;

cout << "sum3 = " << sum(1,2,3) << endl;

return 0;

}

// O/P:

sum1 = 11

sum2 = 11.3

sum3 = 6

2. Operator overloading

- Operator overloading is basically function overloading, where different operator functions have the same symbol but different operands.
- eg:

class Count

{ public:

int a;

int b;

Count (int x, int y)

{

a = x;

b = y;

}

void show()

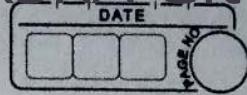
{

cout << "A = " << a << " " << "B = " << b << endl;

}

void operator -()





d

$$a = -a;$$

$$b = -b;$$

3

3;

int main()

d Count obj(20,30);

obj.show();

-obj;

obj.show();

return 0;

3

I/O/P:

$$A = 20 \quad B = 30$$

$$A = -20 \quad B = -30$$

Q) Runtime Polymorphism:-

- Runtime polymorphism is achieved through function overriding.
- The function to be called/invoked is established during runtime.

eg; Virtual function:

- A virtual function is a member function that is declared within a base class and redefined by a derived class.

• eg;

class b

L public:

virtual void show()

d

cout << "In showing base class : ..";

3





void display ()

d

cout << "Displaying base class...";

b

b;

class d : public b

d public:

void display ()

d

cout << "Displaying Derived

class...";

b

void show ()

d

cout << "Showing derived class...";

b

b;

int main()

d

b B;

b * pter;

cout << "P points to base: " << pter;

pter = & B;

pter-> display ();

pter-> show ();

cout << "P points to derived: " << pter;

d D;

pter = & D;

pter-> display ();

pter-> show ();

10/10:

P points to base:

Displaying base class . . .

Showing base class . . .

P points to derived:

Displaying base class . . .

Showing derived class . . .

Q2. Explain the use of keywords. Explain mutable and explicit keywords with examples of each.

Ans. The two unusual keywords: explicit and mutable. mutable have quite different effects, but both are grouped together here because they both modify class members.

- The explicit keyword relates to data conversion, but mutable has a more subtle purpose.

1. mutable Keyword

- mutable keywords come in handy when in a const declared object, you want to update a few const data members without updating other data members.

- eg:

class B

{ public:

 mutable string name;

 int roll

 void nam(string n, int no)



```

    {
        roll = no;
        name = n;
    }

    void changename (string p) const
    {
        name = p;
    }

    void display()
    {
        cout << "Name is: " << name << endl;
        cout << "Roll no is: " << roll << endl;
    }

};

int main()
{
    string p1, p2;
    int rollno;
    b. b;

    cout << "Enter name: ";
    cin >> p1;
    cout << "Enter roll no: ";
    cin >> rollno;
    b. nam(p1, rollno);
    b. display();
    cout << "Enter name again: ";
    cin >> p2;
    b. changename(p2);
    b. display()
    return 0;
}

```



110/p:

Enter name: Tanmay

Enter roll no: 18

Name is: Tanmay

Roll no is: 18

Enter name again: Rajesh

~~Enter~~

Name is: Rajesh

Roll no is: 18

2. Explicit keyword:-

- As discussed in C++, if a class has a constructor which can be called with a single argument, then this constructor becomes conversion constructor because such a constructor allows conversion of the single argument to the class being constructed.
- Prefixing the explicit keyword to the constructor prevents the compiler from using that constructor for implicit conversions.

eg:

Class Blah

{ public:

explicit Blah (int blah)

{

m_blah = blah;

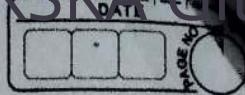
{

int GetBlah ()

{

return m_blah;

{



perovate:

```

int m_Blah;
};

void Ext_Blah (Blah Blah)
{
    int x = Blah. GetBlah ();
}

int main ()
// Your code goes here
Ext_Blah (3);

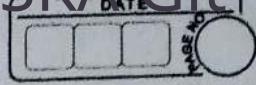
```

- The above code will give the following error:-
 Could not convert '3' from 'int' to 'Blah'
~~Ext_Blah (3);~~
- It is now necessary to call for a conversion explicitly with Ext_Blah(Blah (3)) as shown below:
 int main()
 {
 Ext_Blah (Blah (3));
 }
}

Q3. Explain typecasting concept with example. Also include its types.

- Ans. • Type casting is converting one data type into another one.
- It is also called as data conversion or type conversion.
 - There are two types of type casting operations;
1. Implicit Type Conversion:-
 The type conversion that is done automatically by the compiler is known as implicit type conversion.





- It does not require any effort from the programmer. The C++ compiler has a set of predefined rules.
- Based on these rules, the compiler automatically converts one data type to another.

e.g.:

```
int main()
```

```
{
```

```
    int num;
```

```
    double num1 = 8.28;
```

```
    num = num1;
```

```
cout << "The value of the int variable is: "
```

```
    << num << endl;
```

```
cout << "The value of the double variable is: "
```

```
    << num1 << endl;
```

```
return 0;
```

```
}
```

Output:

The value of the int variable is: 6

The value of the double variable is: 8.28

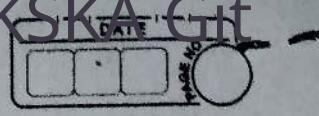
2. Explicit Type conversion:-

- Explicit type conversions are those conversions that are done by the programmer manually.
- In other words, explicit conversion allows the programmer to typecast (change) the data type of a variable to another type.

i) Explicit type C-style type casting:

- This type casting is usually used in the C programming language. It is also known as cast notation.





• Syntax:

(datatype) expression;

• eg:

int main()

{

char char_val = 'a';

int int_val;

int_val = (int) char_val;

cout << "char_val: " << char_val; endl;

cout << "int_val: " << int_val << endl;

return 0;

}

I/O/P:

char_val: a

int_val: 97

2. Function style casting:-

- As the name suggests, we can perform explicit typecasting using function style notations.
- Syntax: datatype(expression);
- eg:

int main()

{

int int_val = 17

float float_val ;

float_val = float(int_val)/2;

cout << "float_val: " << float_val << endl;

return 0;

}

I/O/P:

float_val: 8.5

