KATHMANDU UNIVERSITY

DHULIKHEL, KAVRE

Subject: COMP116

Assignment No: 4

Submitted by:

Ashraya Kadel

CE I/II

Roll no: 25

Submitted to:

Rajani Chulyadyo

Department of Computer Science and Engineering

SUBMISSION DATE: 24/12/2023

Q.1: What is polymorphism? Compare and constrast compile-time and run-time polymorphism.

Ans:

Polymorphism in C++ is an important feature in OOP which is used to define a message or function in many forms.

The differences between compile-time and run-time polymorphism are as follows:

Compile-Time Polymorphism		Run-time Polymorphism
Occurs at compile time	-	Occurs at run-time
It includes function & operator overloading and templates.	-	Achieved through virtual functions and inheritance
It has limited flexibility.	-	It has greater flexibility
It is generally faster.	-	It is comparatively slower.
Used when variations are known at compile type.	-	Used when behaviour can be changed during runtime.
Errors detected at compile type	-	Run-time error is known during execution.

Q.2: What is virtual function? Why do we need virtual functions.

Ans:

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

We need virtual functions for the following reasons:

(i) Ensures correct function is called for an object regardless of the type of reference used.
(ii) Used to achieve run-time polymorphism.
(iii) It is crucial for designing generic and reusable code.
(iv) They allow the extension of functionality make codebase more suitable to changes.
(v) In inheritance, it promotes code orgination.

Q.3: What is virtual destructor? Explain how virtual destructor avoids memory leakage in case of inheritance.

Ans:

Virtual destructor in C++ is a destructor declared with the 'virtual' keyword in a base class. It enables proper destruction of objects in polymorphic hierarchy and helps avoiding memory leaks.

Without virtual destructor, deleting a derived class object through a base-class pointer may skip the call of derived class destructor causing incomplete clean-up.

A virtual destructor is crucial in inheritance as it guarantees deletion of an object.

When object of the 'derived' class is created through 'base' class and deleted, the virtual destructor ensures calling of derived and base class destructor.

This helps to deallocate of any resources used up during object's lifetime.

It ensures destructors are correctly invoked in entire inheritance hierarchy.

Q.4: Differentiate between Interface class and Virtual Base class.

Ans:

The differences between interface class and virtual base class are as follows:

Interface class	Virtual Base Class.
provides contract for classes implementing it.	- Resolves ambiguity in diamond problem
Declares pure virtual functions without any implementation	- May have both virtual and non-virtual functions.
Used to achieve abstraction.	- Used to eliminate redundancy
Acts as blueprint for derived classes.	- Serves as shared base class for derived ~~base~~ class
Leaves the implementation detail to derived class.	- It may include shared functionality that is common.
Doesn't directly address diamond problem	- Directly addresses diamond problem

Q.5: Why do we need to handle exceptions? What is the mechanism in C++ to handle it?

Ans:

We need exception for the following reasons:

i) Gracefully handling errors and implementing recovery strategies.
ii) Providing robustness to a program.
iii) Controlled program termination.
iv) To promote better user experience.
v) Prevent vulnerabilities and security breaches.

In C++, ~~C++~~ exception handling is done through use of try, catch and throw keywords.

In try block, we write code that may cause an exception.

The throw statement throws the exception and the catch block catches the statement. The code written to handle expression is here.

Eg:
```
try
{ int f = factorial(n);
  std::cout<< n<<"!="<<f<<"\n"; }
int factorial(int n)
{ if (n<1){
    throw "n must be greater than zero"}
  int result=1;
  for (int i=1; i<=n; i++) result *= i;
else return result;
}
catch (const char* msg)
{ std::cerr <<"Error: "<< msg <<"\n"; }
```

TQ-64: What do you mean by Generic Programming? Explain Function template and class template.

Ans.

Generic Programming is a programming paradigm that emphasizes writing code independent of datatypes. In this, algorithms and data structures are written in a way that can operate variety of datatypes without sacrifying ~~the~~ performance.

Function templates are special functions that can operate with generic types which serves as a pattern for creating other functions. Eg:

```
template <typename T>
T add (T x, T y)
{ return (x+y); }
add<int> (3,4);
```

Class templates are special classes that can operate with generic types which serves as pattern for creating other classes.

Eg:
```
~~typ~~ template <typename T>
class Pair {
public:
    T first; T second;
    Pair () {}
    Pair (T f, T s): first(f), second(s) {}
};
```