**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | 25-6-2020 | **Name:** | Rasika Patil |
| **Course:** | **C++** | **USN:** | 4AL16EC057 |
| **Topic:** | * Inheritance and polymorphism * Templates, exceptions, and files | **Semester & Section:** | 8th B |
| **Github Repository:** | Rasika B Patil |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **MODULE 5** INHERITANCE AND POLYMORPHISM C:\Users\User\Downloads\WhatsApp Image 2020-06-25 at 8.58.00 PM.jpeg  **C:\Users\User\Downloads\WhatsApp Image 2020-06-25 at 9.01.08 PM (1).jpeg** Inheritance in C++ The capability of a class to derive properties and characteristics from another class is called **Inheritance**. Inheritance is one of the most important feature of Object Oriented Programming. **Sub Class:** The class that inherits properties from another class is called Sub class or Derived Class. **Super Class:**The class whose properties are inherited by sub class is called Base Class or Super class.  **The article is divided into following subtopics:**   1. [Why and when to use inheritance?](https://www.geeksforgeeks.org/inheritance-in-c/#Why%20and%20when%20to%20use%20inheritance?) 2. [Modes of Inheritance](https://www.geeksforgeeks.org/inheritance-in-c/#Modes%20of%20Inheritance) 3. [Types of Inheritance](https://www.geeksforgeeks.org/inheritance-in-c/#Types%20of%20Inheritance)   **Why and when to use inheritance?**  Consider a group of vehicles. You need to create classes for Bus, Car and Truck. The methods fuelAmount(), capacity(), applyBrakes() will be same for all of the three classes. If we create these classes avoiding inheritance then we have to write all of these functions in each of the three classes as shown in below figure: [inheritance](https://media.geeksforgeeks.org/wp-content/uploads/inheritance.png)  The word polymorphism means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.  C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.  Consider the following example where a base class has been derived by other two classes −  #include <iostream>  using namespace std;    class Shape {  protected:  int width, height;    public:  Shape( int a = 0, int b = 0){  width = a;  height = b;  }  int area() {  cout << "Parent class area :" <<endl;  return 0;  }  };  class Rectangle: public Shape {  public:  Rectangle( int a = 0, int b = 0):Shape(a, b) { }    int area () {  cout << "Rectangle class area :" <<endl;  return (width \* height);  }  };  class Triangle: public Shape {  public:  Triangle( int a = 0, int b = 0):Shape(a, b) { }    int area () {  cout << "Triangle class area :" <<endl;  return (width \* height / 2);  }  };  // Main function for the program  int main() {  Shape \*shape;  Rectangle rec(10,7);  Triangle tri(10,5);  // store the address of Rectangle  shape = &rec;    // call rectangle area.  shape->area();  // store the address of Triangle  shape = &tri;    // call triangle area.  shape->area();    return 0;  }  When the above code is compiled and executed, it produces the following result −  Parent class area :  Parent class area :  The reason for the incorrect output is that the call of the function area() is being set once by the compiler as the version defined in the base class. This is called static resolution of the function call, or static linkage - the function call is fixed before the program is executed. This is also sometimes called early binding because the area() function is set during the compilation of the program. Virtual Function in C++ A virtual function is a member function which is declared within a base class and is re-defined(Overriden) by a derived class. When you 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](https://www.geeksforgeeks.org/polymorphism-in-c/) * Functions are declared with a **virtual**keyword in base class. * The resolving of function call is done at Run-time.   **Rules for Virtual Functions**   1. Virtual functions cannot be static and also cannot be a friend function of another class. 2. Virtual functions should be accessed using pointer or reference of base class type to achieve run time polymorphism. 3. The prototype of virtual functions should be same in base as well as derived class. 4. They are always defined in base class and overridden in derived class. It is not mandatory for derived class to override (or re-define the virtual function), in that case base class version of function is used. 5. A class may have [virtual destructor](https://www.geeksforgeeks.org/virtual-destructor/) but it cannot have a virtual constructor.   **Compile-time(early binding) VS run-time(late binding) behavior of Virtual Functions**  Consider the following simple program showing run-time behavior of virtual functions.  filter\_none  edit  play\_arrow  brightness\_4   |  | | --- | | // CPP program to illustrate  // concept of Virtual Functions    #include <iostream>  using namespace std;    class base {  public:      virtual void print()      {          cout << "print base class" << endl;      }        void show()      {          cout << "show base class" << endl;      }  };    class derived : public base {  public:      void print()      {          cout << "print derived class" << endl;      }        void show()      {          cout << "show derived class" << endl;      }  };    int main()  {      base\* bptr;      derived d;      bptr = &d;        // virtual function, binded at runtime      bptr->print();        // Non-virtual function, binded at compile time      bptr->show();  } |   **MODULE 6**  **TEMPLATES, EXCEPTIONS, AND FILES**  **C:\Users\User\Downloads\WhatsApp Image 2020-06-25 at 8.58.00 PM (2).jpegC:\Users\User\Downloads\WhatsApp Image 2020-06-25 at 8.58.01 PM.jpeg** Templates in C++ A template is a simple and yet very powerful tool in C++. The simple idea is to pass data type as a parameter so that we don’t need to write the same code for different data types. For example, a software company may need sort() for different data types. Rather than writing and maintaining the multiple codes, we can write one sort() and pass data type as a parameter.  C++ adds two new keywords to support templates: ‘template’ and ‘typename’. The second keyword can always be replaced by keyword ‘class’.  **How templates work?** Templates are expanded at compiler time. This is like macros. The difference is, compiler does type checking before template expansion. The idea is simple, source code contains only function/class, but compiled code may contain multiple copies of same function/class. [templates-cpp](https://media.geeksforgeeks.org/wp-content/cdn-uploads/gq/2014/06/templates-cpp.jpg)    **Function Templates** We write a generic function that can be used for different data types. Examples of function templates are sort(), max(), min(), printArray(). Know more on [Generics in C++](https://www.geeksforgeeks.org/generics-in-c/)  filter\_none  edit  play\_arrow  brightness\_4   |  | | --- | | #include <iostream>  using namespace std;    // One function works for all data types.  This would work  // even for user defined types if operator '>' is overloaded  template <typename T>  T myMax(T x, T y)  {     return (x > y)? x: y;  }    int main()  {    cout << myMax<int>(3, 7) << endl;  // Call myMax for int    cout << myMax<double>(3.0, 7.0) << endl; // call myMax for double    cout << myMax<char>('g', 'e') << endl;   // call myMax for char      return 0;  } |   Output:  7  7  g  Below is the program to implement [Bubble Sort](http://www.geeksforgeeks.org/bubble-sort/) using templates in C++:  filter\_none  edit  play\_arrow  brightness\_4   |  | | --- | | // CPP code for bubble sort  // using template function  #include <iostream>  using namespace std;    // A template function to implement bubble sort.  // We can use this for any data type that supports  // comparison operator < and swap works for it.  template <class T>  void bubbleSort(T a[], int n) {      for (int i = 0; i < n - 1; i++)          for (int j = n - 1; i < j; j--)              if (a[j] < a[j - 1])                swap(a[j], a[j - 1]);  }    // Driver Code  int main() {      int a[5] = {10, 50, 30, 40, 20};      int n = sizeof(a) / sizeof(a[0]);        // calls template function      bubbleSort<int>(a, 5);        cout << " Sorted array : ";      for (int i = 0; i < n; i++)          cout << a[i] << " ";      cout << endl;      return 0;  } |  Exception Handling in C++ One of the advantages of C++ over C is Exception Handling. Exceptions are run-time anomalies or abnormal conditions that a program encounters during its execution. There are two types of exceptions: a)Synchronous, b)Asynchronous(Ex:which are beyond the program’s control, Disc failure etc). C++ provides following specialized keywords for this purpose.  try: represents a block of code that can throw an exception.  catch: represents a block of code that is executed when a particular exception is thrown.  throw: Used to throw an exception. Also used to list the exceptions that a function throws, but doesn’t handle itself.  **Why Exception Handling?**  Following are main advantages of exception handling over traditional error handling.  ***1)*** Separation of Error Handling code from Normal Code: In traditional error handling codes, there are always if else conditions to handle errors. These conditions and the code to handle errors get mixed up with the normal flow. This makes the code less readable and maintainable. With try catch blocks, the code for error handling becomes separate from the normal flow.  ***2)*** Functions/Methods can handle any exceptions they choose: A function can throw many exceptions, but may choose to handle some of them. The other exceptions which are thrown, but not caught can be handled by caller. If the caller chooses not to catch them, then the exceptions are handled by caller of the caller. In C++, a function can specify the exceptions that it throws using the throw keyword. The caller of this function must handle the exception in some way (either by specifying it again or catching it)  ***3)*** Grouping of Error Types: In C++, both basic types and objects can be thrown as exception. We can create a hierarchy of exception objects, group exceptions in namespaces or classes, categorize them according to types.  **Exception Handling in C++**  **1)** Following is a simple example to show exception handling in C++. The output of program explains flow of execution of try/catch blocks.  filter\_none  edit  play\_arrow  brightness\_4   |  | | --- | | #include <iostream>  using namespace std;    int main()  {     int x = -1;       // Some code     cout << "Before try \n";     try {        cout << "Inside try \n";        if (x < 0)        {           throw x;           cout << "After throw (Never executed) \n";        }     }     catch (int x ) {        cout << "Exception Caught \n";     }       cout << "After catch (Will be executed) \n";     return 0;  } |   File Handling In C++  Files are used to store data in a storage device permanently. File handling provides a mechanism to store the output of a program in a file and to perform various operations on it.  A stream is an abstraction that represents a device on which operations of input and output are performed. A stream can be represented as a source or destination of characters of indefinite length depending on its usage.  In C++ we have a set of file handling methods. These include ifstream, ofstream, and fstream. These classes are derived from fstrembase and from the corresponding iostream class. These classes, designed to manage the disk files, are declared in fstream and therefore we must include fstream and therefore we must include this file in any program that uses files.  In C++, files are mainly dealt by using three classes fstream, ifstream, ofstream.   * ofstream: This Stream class signifies the output file stream and is applied to create files for writing information to files * ifstream: This Stream class signifies the input file stream and is applied for reading information from files * fstream: This Stream class can be used for both read and write from/to files.   All the above three classes are derived from fstreambase and from the corresponding iostream class and they are designed specifically to manage disk files. C++ provides us with the following operations in File Handling:   * Creating a file: open() * Reading data: read() * Writing new data: write() * Closing a file: close()   Moving on with article on File Handling in C++  Opening a File  Generally, the first operation performed on an object of one of these classes is to associate it to a real file. This procedure is known to open a file.  We can open a file using any one of the following methods: 1. First is bypassing the file name in constructor at the time of object creation. 2. Second is using the open() function.  To open a file use   |  |  | | --- | --- | | 1 | open() function |   Syntax   |  |  | | --- | --- | | 1 | void open(const char\* file\_name,ios::openmode mode); |   Here, the first argument of the open function defines the name and format of the file with the address of the file.  The second argument represents the mode in which the file has to be opened. The following modes are used as per the requirements. |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |