W3101 Programming Languages – C++ Midterm exam Oct 01, 2008

Name: Student Id:

---

1. Explain the following (OOP) features briefly and show how they are implemented in C++, using ONE example. That is, use a single example (say a base class and a child class) to explain all the features. . . . (4 marks)

   Example:

```
class parent                          class child : public parent
  {                                     {
    private:                             public:
      int a, b;                            child() {/*code*/}
    protected:                            ~child() {/*code*/}
      int p;                               void f1() {/* code */}
    public:                                void f2() {/*code*/}
      int x;                               void f2(int i) {/*code*/}
      int getA() { return a; }             void f3(const int& j);
      parent() {/* code */}             };
     ~parent() {/* code */}
      virtual void f1() = 0;
      virtual void f2() {/* code */ };
   };
```

   (a) Data encapsulation
   Answer: Data encapsulation is a feature of where the data is hidden from the user and the access to read (or modify) is provided through class methods (or functions). The advantage of this featue is that the implementation details of a method are hidden from an end user of a class. It is possible to change the internal implementation of a method in a class without changing the way the method itself is invoked. In C++ "private", "protected" or "public" define the various access restrictions for class members.

   (b) Inheritance with public, private and protected members
   Answer: Inheritance is a feature where one class is designated as a "parent" class and some other class is designated as a "child" class. The members (data and functions) of a parent are made available to the child class. There are many advantages of inheritances, cod re-use being one of the main ones. Access restrictions of data members are as follows:

   public: Accessible to any external class or function.
   protected: Accessible to friends and child classes.
   private: Accessible only to friends.

(c) Polymorphism
Answer: In a literary sense, “poly” means “many” and “morphism” means “form” or “shape”. Polymorphism in C++ is a feature where an object can have different forms or shapes (as a base class or one of the derived classes), depending on how it is used. object of “parent” or “child” depending on how it can be used. In C++ polymorphism can be achieved using virtual functions (see below). An object of class “parent” can behave as an

(d) Virtual functions and pure virtual functions
Answer: In C++ it is possible to assign the pointer to a base class to the address of a derived class (*e.g.*, `base *obj = new derived;`). Using the “virtual” keyword, we can force the object to call the class specific function. In the above example, `obj` can point to an object of a different derived class each time and exhibit the behavior of that class. Or, it can also behave as a base class (if it indeed points to an object of base class). For example, `obj` can behave as a generic (base) class called “employee” or it can behave as a specific (derived) class called “executive” or “officer” or “researcher”, etc.
Pure virtual function is one for which there is no body in the base class; it is set to 0. Derived classes implement the code for pure virtual functions. In the example above, “f1()” is a pure virtual function, while “f2()” is a (non-pure) virtual function.

(e) Abstract classes
Answer: Any class that has atleast one pure virtual function is an abstract class. Objects of an abstract class cannot be instantiated. In the example above “parent” is an abstract class.

(f) Function overloading
Answer: Two or more functions that have the same name but different arguments are overloaded functions. In the above example, “f2(..)” is an overloaded function in class “child”.

(g) Passing parameters by reference
Answer: The changes done inside a function to a parameter passed by reference are reflected outside the function, after the function exits. In the example above, “j” is passed as a reference to function “f3()”. Any changes done to “j” inside “f3()” remain valid after “f3()” returns.

(h) Constructor and Destructor
Answer: Constructor is the code that is called when an object is instantiated. It is a function with the same name as the class. Typically, member variables are initialized in the constructor and memory allocation to any pointers is done. A class can have more than one constructor. Destructor is the opposite of a constructor. It is a function with the same name as the class, with a “ ” prefix. It is called when an object goes out of scope. Typically, resource cleanup, e.g., freeing allocated memory, is done in the destructor. In the example above “child()” is the constructor and

“ child()” is the destructor of the class “child”.

2. Answer “True” or “False” to the following questions with explanation. . . . (2 marks)

   (a) C++ programs form a superset of C programs in terms of the syntax, declarations and language specific constructs.

      Answer: True, C++ syntax is a superset of C. Any C program that can, in theory, be compiled by a C++ compiler.

   (b) Complex C++ programs can not logically be implemented in C programming language because OOP specific features and other C++ related features are not supported in C language.

      Answer: False, any C++ program logic can be implemented in C also, (or for that matter in any other programming language). The syntax and logic would be more complex in C than they would be in C++.

3. Consider the following code segment: . . . (3 marks)

```
class baseClass
{
  public:
    baseClass()
      { cout << ‘‘In base class constructor’’ << endl; }
   ~baseClass()
      { cout << ‘‘In base class destructor’’ << endl; }
    virtual void f1()
      { cout << ‘‘In base class f1’’ << endl; }
    void f2() { cout << ‘‘In base class f2’’ << endl; }
};

class derivedClass
{
  public:
    derivedClass()
      { cout << ‘‘In derived class constructor’’ << endl; }
   ~derivedClass()
      { cout << ‘‘In derived class destructor’’ << endl; }
    void f1() { cout << ‘‘In derived class f1’’ << endl; }
    void f2() { cout << ‘‘In derived class f2’’ << endl; }
};
```

What is the ouput of the following program segment? Please write your answers next (or below) to the functions called in main.

```
  main()
  {
    baseClass x;
// Answer: In base class constructor

    derivedClass y;
// Answer: In base class constructor
//         In derived class constructor

    baseClass *z = new derivedClass;
// Answer: In base class constructor
//         In derived class constructor

    x.f1();  // Answer: In base class f1}

    x.f2();  // Answer: In base class f2}

    y.f1();  // Answer: In derived class f1}

    y.f2();  // Answer: In derived class f2}

    z->f1(); // Answer: In derived class f1}

    z->f2(); // Answer: In base class f2}

// Answer: In base class destructor    // x goes out of scope
//         In derived class destructor // y goes out of scope
//         In base class destructor    // y goes out of scope
 }
```

4. A virtual destructor similar to a virtual function, except that instead of any function, the destructor is made virtual. The behavior of a virtual destructor is similar to that of any virtual function. Show with an example how a virtual destructor can be implemented. Why would anyone need to implement a virtual destructor? Does it serve any purpose to have a virtual destructor, as opposed to having a regular, non-virtual destructor? . . . (3 marks)
Answer: A virtual destructor is used for the same reason as any virtual function is used, to ensure that the class specific destructor is called. Consider the following code segment

```
     class baseClass
     {
```

```
  public:
    baseClass() { }
   virtual ˜baseClass() { }
}

class derivedClass : public baseClass
{
   int *i;
   derivedClass() { i = (int *) malloc (sizeof (int)) ; }
  ˜derivedClass() { free (i); }
}

main()
{
  baseClass *a = new derivedClass;
  delete (a);
}
```

Here if the destructor is not virtual, the base class destructor is called, causing a memory leak. By using a virtual destructor, the derived class destructor is called first, followed by a base class destructor, cleaning up all the resources.

5. Multiple inheritance is a case where one class is derived more two or more classes. For example, the following declaration shows that square is derived from both shape and polygon.

   ```
   class square : public shape, polygon
   ```

   What are the advantages of multiple inheritance? What are the negative points and potential problems of multiple inheritance? Support your arguments with examples for both advantages and disadvantages. . . . (3 marks).

   Answer: The advantages of multiple inheritance is that, we don't have to rewrite the code of the base classes all over again in the derived class. Derived class has access to the public members of all the base classes it is derived from.

   The disadvantages are that, there can be clashes in the data of the parent classes. In the above example, we have to make sure that `shape` and `polygon` do not have any membes with the same name. Another problem is the case where `class p` and `class q` are each derived from `class a` and `class x` is derived from both `q` and `q`. This is the standard "diamond" structure. This can create ambiguities in programming and care has to be taken to resolve them.