John Pennig

Homework 11

**Q 1. Explain the general subprogram characteristics and define the basic terms associated with subprograms.** General characteristics is each subprogram has a single entry point, only one subprogram can execute at a given time, and control is always given back to the calling function when execution terminates.

A subprogram definition describes the interface to and the actions of the subprogram abstraction.

A subprogram call is the request that the subprogram be executed.

A subprogram header is the first part of the definition.

**Q 2. Explain the terms actual parameters, formal parameters, positional parameters and keyword parameters with examples.**

Actual Parameters: Actual parameters are the parameters in the subprogram call. An example is sum(num1, num2) where num1 and num2 are actual parameters.

Formal Parameters: These are parameters that are in the subprogram header. An example is if you bound a parameter to a variable such as def count(sum) where sum is a formal parameter.

Positional Parameters: Positional parameters are the relation of actual parameters to formal parameters. An example is the first actual parameter is bound to the first formal parameter and so forth.

Keyword Parameters: Keyword parameters use the actual name of the formal parameter assigned to an actual parameter to allow the user to put the actual parameters in any order. An example would be count(sum = my_sum, current = current_value).

**Q 3. What are the different semantic models of parameter passing? Explain with examples.**

In mode: While in in mode, the formal parameters can receive data from the corresponding actual parameters.

Out mode: While in out mode, the formal parameters can transmit data to the corresponding actual parameters.

Inout Mode: While in inout mode, they can do both.

An example of this would be a subprogram that takes in two lists, list1 and list2, and returns a revised list of list2. List1 should be in in mode because it will not be changed at all during execution, list2 should be in inout mode because it will be changed and must send its values out. The new list2 should be in out mode since it will be modified and revised.

**Q 4. What are the implementation models of parameter passing? How are they related to the semantic models? Explain.**

Passed-by-value: The value of the actual parameter is used to initialize the corresponding formal parameter. This acts as a local variable in the subprogram which enables in-mode semantics.

Pass-by-result: Pass-by-result is an implementation model for out-mode parameters. When a parameter is passed by result, no value is transmitted to the subprogram.

Pass-by-value-result: Pass-by-value-result is an implementation model for inout-mode parameters in which actual values are copied. It is a combination of the previous two. The value of the actual parameter is used to initialize the formal parameter which then acts as a local variable.

Pass-by-reference: Another an implementation model for inout-mode parameters. Pass-by-reference sends an access path, usually just an address to the called subprogram. This gives the subprogram the access path to the cell storing the actual parameter.

Pass-by-name: An inout-mode parameter transmission method that does not correspond to a single implementation model. When parameters are passed by name, the actual parameter is textually substituted for the formal parameter in all its instances in the subprogram.

**Q 5. Explain the design issues for functions.** The design issues for functions include are side effects allowed, what types of values can be returned, and how many values can be returned. Side effects can be a major design issue for functions to make sure that the program is performing the intended effect. The values that can be returned from a function can vary by language, most restrict the type that can be returned but others such as C and C++ offer more freedom. Most languages only allow one value to be returned, but others have tuples that can package more than one value.

**Q 6. Chapter 9 Problem set: Question 6. Present one argument against providing both static and dynamic local variables in subprograms.**

An argument against both static and dynamic local variables in subprograms is the complexity. Understanding and reading the code with both static and dynamic local variables can be hard to understand and can cause confusion. This can lead to programming errors and hard to read code.

**Q 7. Chapter 11 Review Questions: 1 – 6, 21, 24, 26, 28, 29.**

1. What are the two kinds of abstractions in programming languages? Two kinds of abstractions in programming languages are subprograms and Classes.

2. Define *abstract data type*. An ADT is a data structure, in the form of a record, but which includes subprograms that manipulate the data.

3. What are the advantages of the two parts of the definition of *abstract data type*? An advantage is the packaging of data objects with their associated operations. Another advantage is information hiding which can increase reliability since the ADT can only be manipulated by the operations defined by the programmer.

4. What are the language design requirements for a language that supports abstract data types? The language must provide a syntactic unit that encloses the declaration of the type and of the prototypes of the subprograms that implement the operations of on objects of the type.

5. What are the language design issues for abstract data types? Language design issues include whether ADT's can be parameterized, what access controls are

provided and how are such controls specified, and whether the specification of a type is physically separate of its implementation.

6. From where are C++ objects allocated? C++ objects are allocated either static, stack-dynamic, or heap-dynamic.

21. Explain the three reasons accessors to private types are better than making the types public. Access control, this limits access to these private types to ensure that they are not changed by accident. Information hiding, to ensure that the information is not accessed by different methods or different objects. The definitions are hidden from clients because the client do not need to know how or what a private type is.

24. What is the fundamental difference between the classes of Ruby and those of C++ and Java? In Ruby, everything is an object, and arrays are actually arrays of references to objects.

26. Describe the two problems that appear in the construction of large programs that led to the development of encapsulation constructs. When programs are so large, there is not enough organization to make it reasonable to modify further. Another problem is recompilation, recompiling large programs can be very costly.

28. What is a C++ namespace, and what is its purpose? A C++ namespace is a logical encapsulation of names that define the scope of that name. It is a way to package names into a specific library to help manage the global namespace.

29. What is a Java package, and what is its purpose? A Java package is another naming encapsulation construct. Its purpose is to contain more than one type definition and the types in the package are partial friends to each other meaning types that are not private are visible to all others in a package.

**Q 8. How are Interfaces and Abstract classes defined in Java? Why are they useful? How are these related to the concept of data abstractions, and how are they different from concrete classes?** Interfaces are defined using the interface keyword such as public interface name{}. Abstract classes are defined using the abstract keyword such as abstract class name{}. Interfaces and Abstract classes are both ways to specify type definitions without having to have concrete definitions. They differ from concrete classes because interfaces can be used to only specify method signatures, and abstract classes can be used if we don't know all the methods that are needed for a concrete class yet.

**Q 9. Explain with an example how Java deals with run-time errors.** Run-time errors are managed by exceptions in most object-orientated languages. Java handles exceptions with try-catch blocks to attempt to handle the error. An example could be if a certain function expected a double as a parameter type and the program passed along a type of int, the try block could identify the wrong type and the catch block could convert the int to a double.

**Q 10. Explain the terms Inheritance, Polymorphism and Dynamic Binding with examples.**

Inheritance is the ability of a class to pass down its methods to another class. The ancestor or base class passing down its attributes to its descendent or derived class. An example would be a class a Motor class that passes down attributes to a Car class since all cars have motors.

Polymorphism relates to anything that can take on multiple forms. Such an example could be a class Vehicle that can either take on Car, or Truck classes. Since a vehicle can be either a car or a truck.

Dynamic Binding is associated with polymorphism and allows us to invoke methods on members of a class hierarchy without knowing the specific object. An example of dynamic binding using the Vehicle, Car, and Truck classes could be if you had an object Chevrolet of Class Truck and you called a method from Truck. The Dynamic Binding would be the process of choosing where that method comes from whether it was from Truck or Vehicle.

**Q 11. Explain why polymorphic assignments result in a loss of information. In spite of this loss of information, why are they useful, and what language features enable their usefulness. Explain with examples.** Polymorphic assignments result in a loss of information because we store a subclass as a type of the superclass causing us to no longer be able to access the subclass members while using the superclass reference. Dynamic binding however allows us to enable their usefulness by looking into the contents of the object to determine what type it

is, and choose the correct methods for that object. An example from the book was to put fruits in a box on display, and the worker looks at the box and determines the fruit and the correct display method.

**Q 12. What is the Object class in Java? Why is it useful? From your experience with C++, do you feel that such a class would be useful in C++? Explain.** The Object class in Java is a special class that every single other class inherits from. If you declare a reference to be of type Object, then you can store a reference to any object. No, I do not think that such a class would be useful in C++, the main advantage of C++ over other languages is the speed and this would just add to execution time.

**Q 13. Explain with a simple example how Java provides support for generic classes.** Generic classes are classes that have parameters that do not have a specific type and are not concretely defined. They have generic parameters which are placeholders for the parameter types. When the class is created, the generic parameters must be replaced by an actual type. If you have a Vehicle class that must store different types of vehicles such as car types and truck types, you can give it a generic type and then if you attempt to put a car type in a truck type stack of the vehicle class, it will throw an error.

**Q 14. What is the Java event model? How is it used to deal with mouse clicks and button clicks?** The Java event model is the model for how Java deals when a user interacts with a GUI component. When a user clicks the mouse or a button, the component creates an event object that calls an event handler through another object called the event listener which passes the event object on.