HOMEWORK 3

Data types and OOP

Oscar Galindo, Russell Gehan

1. (5 points) What are advantages and disadvantages of decimal data types?

Advantages: accuracy

Disadvantages: limited range, wastes memory

2. (10 points) Write a short discussion of what was lost and what was gained in Java's designers' decision to not include the pointers of C++.In C++, pointers allow the program to pass arguments by reference.

Pointers can be used to create aliases, and they're especially useful when accessing arrays via pointer offsets. However, C++ user often had to build wrapper classes for arrays to add features like index range checking. In Java, arrays are a class of their own and features, like index range checking, are already included. Even though Java doesn't have pointers, it can still create reference types and pass objects in other ways. Java also allows its references to be allocated on the heap and dereferenced implicitly, which can save memory. Furthermore, pointers are insecure because they can point to instances that do not exist anymore.

3. (10 points) What are the arguments for and against Java's implicit heap storage recovery, when compared with the explicit heap storage recovery required in C++?

Advantages over C++: It is automatic reclamation of storage with the consideration of lifetime of objects, prevents the creation of dangling pointers and the heap fragmentation by handling removed objects in such a way they become freed space for other, still existing objects.

Disadvantage: Impacts performance due to continuous checking of objects stored throughout memory and it is impossible to predict for how long and when will the implicit recovery system will execute, whereas in C++ the reclamation process is manual but very predictable.

4. (10 points) Multidimensional arrays can be stored in row major order, as in C++, or in column major order, as in Fortran. Develop the access functions for both of these arrangements for three-dimensional arrays.
L = Address of array
S = Size of each variable
m = numbers of cols
n = number of rows
p = number of layers
Row Major: f(i, j, k) = L + S*(i*n*p + j*p + k)
Column Major: f(i, j, k) = L + S*(j*m*p + i*p + k)

5.(10 points) Make two lists of applications of matrices, one for those that require jagged matrices and one for those that require rectangular matrices. Now argue whether just jagged, just rectangular, or both should be included in a programming language.

Jagged:
-Efficiency improved models over Rectangular arrays (unused entries are not included).
-Classification of objects or storage of values is not symmetric, but instead unregular in distribution, for example a calendar.
-Any unprecise (in terms of dimensions length) configuration of arrays.

Rectangular:
-Matrix/vector operations (the row major or column major orders are suitable for this operations).
-Editing Distance operations
-Any precise (in terms of dimensions length) configuration of arrays.

In our opinion both ragged/jagged arrays and rectangular arrays should be part of any programming language. In the case of ragged arrays this kind of arrays allows us to reduce the space complexity of any arrangement because we are able to no utilize the arrangements we do not need, whereas in the rectangular kind of arrays we must utilize entries that might not be needed. The reduction of space complexity, the locality of the objects/values stored, after the first entry of every array in the array of arrays is located, permit a reduction an efficient use of memory space. In comparison, multidimensional arrays are fundamental for matrix operations and any other comparable arrangements. The fact that multidimensional arrays can be arranged in row major and column major orders allow us to easily access and retrieve values from the possibly huge arrangement in constant time. This is powerful when the computational goal is to collapse or produce a result from the arrangement represented by the multidimensional array. This is due to the fact computationally all positions and memory locations can be retrieved by using: f(i, j, k) = L + S*(j*m*p + i*p + k) or f(i, j, k) = L + S*(i*n*p + j*p + k).

6. (10 points) Explain the advantages or disadvantages of having all values in a language be objects.

The advantage of this choice is the elegance and pure uniformity of the language and its use. The primary disadvantage is that simple operations must be done through the message-passing process, which often makes them slower than similar operations in an imperative model, where single machine instructions implement such simple operations. In this purest model of object-oriented computation, all types are classes. There is no distinction between predefined and user-defined classes. In fact, all classes are treated the same way and all computation is accomplished through message passing.

The advantage of having all values in a language be objects is having uniformity within the language and in the way it's used. All objects have local memory, inherent processing ability, the capability to communicate with each other, and the capability of inheriting methods and variables from super classes. The disadvantage is that all simple operations have to be done through message-passing processes, which can be slower than systems performing simple operations on primitive types

7.(10 points) Summarize the fundamental argument for dynamic method binding. Why do C++ and C# use static method binding by default?

The argument for dynamic method binding is that when we have a collection of subclasses that emanate from the same class, usually those classes have similar methods, suppose a shape superclass and circle, oval, square, and rectangle subclasses each contain a draw method. In order to decide which of the possible draw methods to use throughout the inheritance relationships dynamic method binding is needed. This means the main idea of dynamic method binding is that dynamically, or as the context changes, the methods referenced used by the object along its inheritance relationship(s) change.

The reason static method binding is a default in both languages is because most of the methods implemented in classes are not meant to be overridden, if the methods are meant to be overridden then both languages require you to indicate it. Also, a second concern might be that if a language like C++ allows multiple inheritance, then even if dynamic binding is activated it is not clear which of two equivalently named and inherited methods will be run from a specific subclass (diamond problem).

8. (5 points) What is one programming situation where multiple inheritance has a significant advantage over interfaces?

A class can be derived from a super class and can also implement an interface at the same time. This is sometimes used to simulate multiple inheritance. However, interfaces don't provide reusable code like inherited classes do. Multiple inheritance then is especially useful when you want to build a class that can reuse several variables and methods from multiple super classes.

9.(5 points) Explain why allowing a class to implement multiple interfaces in Java and C# does not create the same problems that multiple inheritance in C++ creates.

In C++ multiple inheritance can create the conditions of the diamond problem, in which a subclass implements two superclasses, but it might be that these two super classes contain a method that is equal in terms of the method signature. If the subclass were to make a call of this doubly-referenced method, then at the compilation stage of the program an error will show saying more or less: "member '[name of method]' found in multiple base classes of different types" at which point the developer will have to indicate which of the two or more implementations does the developer is referring to.

In contrast, Java only allows for multiple inheritance of interfaces at which point even if the two interfaces were to require the inclusion of a method with the same signature the implementation will be made at the scope of the subclass and not in the inheritance scope. C#, on a similar idea, follows Java implementation where interfaces can be implemented multiple times, but again, the possible problem issue with diamond problem does not occur for the same reasons as in Java.

10. (10 points) Describe the issue of how closely the parameters of an overriding method must match those of the method it overrides. Consider the rule of Java 8.

In Java, a method must have the same signature of the method that it's overriding. This means that this method must be passed the same parameters as the overridden method, and it must return the same type as the overridden method. The overriding method also cannot throw new exceptions not already thrown by the overridden method. Finally, static and final methods cannot be overridden.

11. (15 points) Rewrite the following C++ classes in Java and compare the result with the C++ version in terms of readability and writability. Hint: Both stack_2 and queue_2 are *private* subclasses.

Java Code:

```
////////////////////////////////////////////////////////////////////////////
class single_linked_list {
   private class node {
        public node link;
        public int contents;
        };
   node head;
   public single_linked_list() {head.contents=0};
   public void insert_at_head(int);
   public void insert_at_tail(int);
   public int remove_at_head();
   public boolean empty();

}

////////////////////////////////////////////////////////////////////////////

private class stack_2 {
private single_linked_list list;
public stack_2()
{
  list = new single_linked_list();
}
void push(int value) {
              list.insert_at_head(value);
}
int pop() {
  if(!list.empty())
              return list.remove_at_head();
  else
              return null;
}
}
```

```
////////////////////////////////////////////////////////////////////////////

private class queue_2{
   private single_linked_list list;
   public queue_2()
   {
     list = new single_linked_list();
   }
   void enqueue(int value)
   {
     list.insert_at_tail(value);
   }
   int dequeue()
   {
     if(!list.empty())
              return list.remove_at_head();
     else
              return null;
   }

}

////////////////////////////////////////////////////////////////////////////
```

After the conversion process, it can clearly be seen that the code needed to express an equivalent class and subclass arrangement in Java is much longer. This means that writability when compared between C++ and Java is in favor of C++. Readability, for the consequences of having a concise set of statements, is also in favor of C++. In general, if a programming language requires a smaller set of statements to express a desired arrangement of classes and subclasses it is not only easy to write but also easy to understand, hence, readability and writability are higher in C++ for this specific purpose.

12. (10 bonus points) Explain subtyping for Java 5 Generic (parameterized) classes? I.e., when an instantiated class is a subtype of another instantiated class and why?

In Java 5.0, the reserved word “extends” indicates whether a class is a subtype of another class. A generic class is specified when its name is followed by at least one type variable enclosed in pointed brackets. ArrayLists, Stacks, and LinkedLists are some common examples of generic classes. The type variable enclosed in the pointed brackets of a generic class can be bounded as a subtype of another class if it’s followed by the words “extends “and the class type. In this manner, programmers can ensure that the elements of the instantiated generic class are bound to an interface or super class.