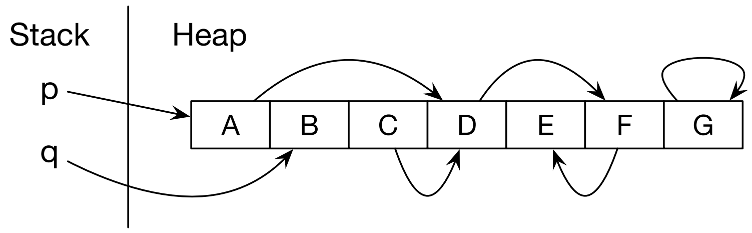
CMPSC 461: Programming Language Concepts
Assignment 6 Solution

Problem 1 [12pt] Consider the heap state as shown below before garbage collection:



a) (3pt) For the Mark-and-Compact algorithm, what objects remain on the heap after collection? Do they stay in the same location after collection?

   Solution:
   Objects A, B, D, E, F remain on the heap after collection. No.

b) (9pt) Consider algorithms Mark-and-Sweep, Mark-and-Compact, Stop-and-Copy. For each of them, write down the objects being visited during the collection in sequence. Assume 1) reachability analysis uses depth-first search, which will skip objects that have already been visited, 2) search starts from p, and then q, 3) when the entire heap is traversed, objects are visited from left to right. You don't need to write down the newly created object copies, if any.

   Solution:
   Mark-and-Sweep: A, D, F, E, B (Mark), A, B, C, D, E, F, G (Sweep).

   Mark-and-Compact: A, D, F, E, B (Mark), A, B, C, D, E, F, G (Compute new addresses), A, B, C, D, E, F, G (Move objects to new address).

   Stop-and-Copy: A, D, F, E, B.

Problem 2 [8pt] For each of the following data characteristics, describe which of the garbage collection algorithms discussed in class (Mark and Sweep, Mark and Compact, Stop and Copy) would be the most appropriate. Make no other assumptions than those given.

a (4pt) All objects are small in size and most have relatively short lifetime.

   Solution:
   Stop and copy is the best since most objects have short lifetimes. The collection time of stop and copy is proportional to live objects. Also, it eliminates fragmentation.

b (4pt) Most objects have long lifetime and data locality greatly improves application's performance.

   Solution:
   Mark and compact is the since most objects have long lifetimes. Mark and compact eliminates fragmentation, and hence, improves data locality.

Problem 3 [5pt] Assume you're required to implement an Abstract Data Type (ADT) for a checking account. The requirement is that the ADT should provide operations such as deposit, withdraw and check balance. Since this is a checking account, you are required to enforce an invariant that the balance is never negative. With this invariant in mind, state a contract (in terms of precondition, postcondition and side effects) on the "withdraw" operation: `public void withdraw (double amount)`.

Solution:One possible solution
Precondition: amount ≥ 0
Postcondition: deduct input amount to balance when amount<balance; no effects otherwise
Side effects: the operation changes current balance

Problem 4 [10pt] Consider the following Java classes:

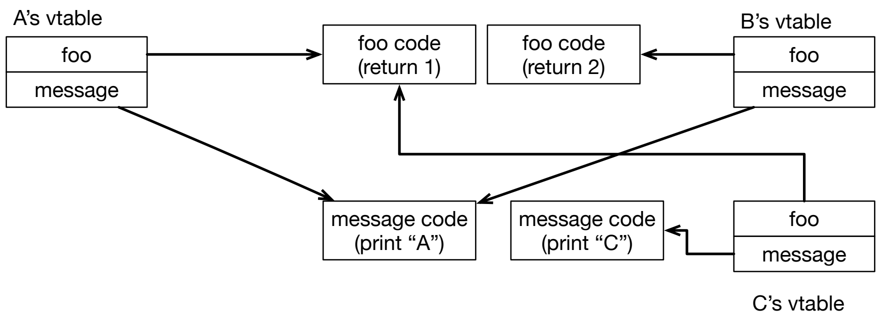
```
class A {
    public int foo () { return 1;}
    public void message () { System.out.println( "A"); }
}

class B extends A {
    public int foo() {return 2;}
}

class C extends A {
    public void message () { System.out.println( "C"); }
}
```

a) (5pt) Draw the virtual tables of class A, B and C, as well as the code implementations that they point to.

Solution:



b) (5pt) Consider the following function:
`void func(C c) { c.message(); }`

With subtyping polymorphism, this function prints "A" given an instance of class A; prints "C" given an instance of class C. Give the pseudo code for the body of `func` after compilation to explain how does this happen with dynamic dispatching. Assume that the base address of `c` is stored in a pointer `this`, and that the target machine has 4-byte addresses. You can ignore the calling sequence (e.g., save and restore registers, store return address and so on).

Solution:
The compiled code looks like:

```
vt = *this; // find the vtable from the object's base address
call *(vt+4); // vt+4 is the address of function "message"
```

Problem 5 [15pt] Below is the pseudo code that computes $x + 3 + 5 \times y$. There is a complete Hoare triple for the loop, but no precondition for the entire program.

```
{        }
z = x;
z = z+3;
n = y;
{$n \geq 0 \wedge z = x + 3 \wedge n = y$}
while (n>0) {
  z = z + 5;
  n = n - 1;
}
{$z = x + 3 + 5 \times y$}
```

a) (5pt) Complete the following Hoare triple by writing down the weakest precondition for the first three statements in this program fragment. You can simply your postcondition in the solution.

```
{       }
z = x;
z = z+3;
n = y;
{$n \geq 0 \wedge z = x + 3 \wedge n = y$}
```

Solution:

$$y \geq 0$$

b) (10pt) Write down a loop invariant to prove the correctness of this program, and briefly explain why the invariant is correct. That is, write down an invariant, and briefly explain why it is 1) true before the first execution of the loop, 2) true after the execution of each loop iteration, given that the loop condition and the invariant are both true before the iteration, and 3) strong enough to prove the correctness of the code (i.e., $z = x + 3 + 5 \times y$ is true after the loop, if it terminates).

Solution:
The invariant is $n \geq 0 \wedge z = x + 3 + 5 \times (y - n)$. We use $\mathcal{I}$ to represent this loop invariant.

Condition 1): easy to check that $n \geq 0 \wedge z = x + 3 \wedge n = y$ implies $\mathcal{I}$.

Condition 2): we need to check the following Hoare triple is true:

```
{$n > 0 \wedge \mathcal{I}$}
z = z + 5;
n = n - 1;
{$\mathcal{I}$}
```

Based on the postcondition, we can compute that the weakest precondition is

$$n - 1 \geq 0 \wedge z + 5 = x + 3 + 5 \times (y - (n - 1))$$

which can be simplified as

$$n \geq 1 \wedge z = x + 3 + 5 \times (y - n)$$

So the Hoare triple above is correct since $n > 0 \wedge \mathcal{I}$ implies $n \geq 1 \wedge z = x + 3 + 5 \times (y - n))$ (n is an integer).

Condition 3): Easy to check that $n \leq 0 \wedge \mathcal{I}$ implies the postcondition $z = x + 3 + 5 \times y$.