

Homework 6

Due: 11 Nov 2009 1:55pm

All homeworks are due at 1:55pm in the **CS 31 bin** on the second floor. No late homeworks are accepted.

Please include your *login name* on each piece of paper you hand in, and please staple your pages together before handing in.

Some of the problems on this homework must be handed in electronically. Please do each problem in a separate file. Run `cs031_handin hw06` in the directory where your files are stored to hand in. You will be prompted for the name of each file.

Please do the first two problems on paper and the third electronically.

Problem 6.1

- a. In lecture, Pascal made the bold claim that the first fit allocation scheme is more robust. He is correct. He is always correct.

He does note in passing that there are, in fact, counterexamples to his claim.

Give one such example; that is, find a case in which first fit allocation would get “stuck” before best fit allocation.

- b. Compaction and fixing fragmentation are two very different deletion algorithms. On the next page you are given a diagram representing the state of memory after marking. Perform compaction on this diagram of memory. Show your work as follows:
- Phase 1: Label the forwarding address of each applicable block in the left diagram.
 - Phase 2: Correct any pointers as necessary by crossing out the old address in the left diagram and writing the new one in its place.
 - Phase 3: Fill in the right diagram so it reflects how the memory will look after the whole process completes.

0	marked: no size: 4 forwarding: pointer to 36	0	
4	marked: yes size: 5 forwarding: pointer to 20 pointer to 36	4	
8	marked: no size: 8 forwarding: pointer to 0	8	
12	pointer to 20	12	
16		16	
20	marked: yes size: 3 forwarding:	20	
24	marked: yes size: 11 forwarding: pointer to 17	24	
28	pointer to 20 pointer to 4	28	
32	marked: no size: 5 forwarding: pointer to 9	32	
36	marked: yes size: 5 forwarding: pointer to 17	36	
40	marked: no size: 2 forwarding:	40	

- c. In the C language, pointers can be stored in integer variables and manipulated as unsigned (or even signed) integers. At any time, any integer can be converted to and used as a pointer. (Assume we have 32-bit pointers and 32-bit integers). What major challenge does allowing this practice pose to compaction? Would fixing fragmentation face the same problem? Why or why not? Please be specific in your answers.

Problem 6.2

Which of the following Garbage Collection algorithms will successfully find all garbage nodes in the following graphs (a, b, and c)? Keep in mind that Garbage Collection is done ‘periodically’ (from time to time), and not ‘continually’ (every time a node is referenced or dereferenced).

Your choices are

- (1) Bellatrix Lestrange’s “Dark Mark” reference counting (aka Mark and Sweep), as shown in lecture.

This algorithm starts from the root node and marks all nodes it traverses. Next, unmarked nodes are swept up (put back on the free list).

- (2) Snape and Draco’s “Unbreakable Vow” reference counting.

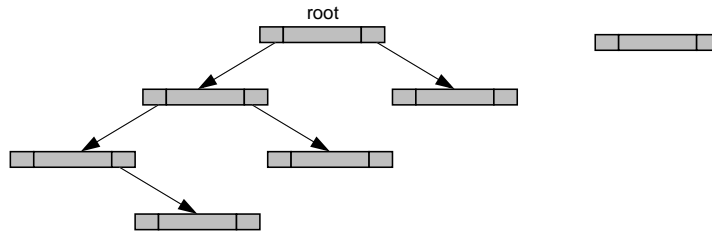
In Alex’s reference counting, we keep a counter for each node. Every time a node is referenced, its counter is incremented. Every time a node is unreferenced, its counter is decremented. During Garbage Collection, nodes with counters of 0 are disposed of. At the end of Garbage Collection, nodes referred to by the disposed nodes have their reference counts lowered; nothing else happens.

- (3) McGonagall’s “Ten Points from Gryffindor” reference counting.

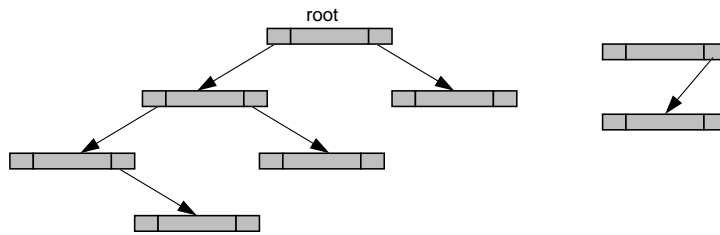
This algorithm keeps a reference count like the Simple reference counting algorithm. During garbage collection, nodes with counters of 0 are disposed of. Nodes referred to by the disposed node have their reference counts decremented. If this lowers a node’s reference count to 0, that node is also garbage collected.

For each of the following pictures, list whose algorithms will successfully collect all garbage nodes in one run. Each box represents a node in memory with two pointers to other nodes, as in the Garbage Collection lecture.

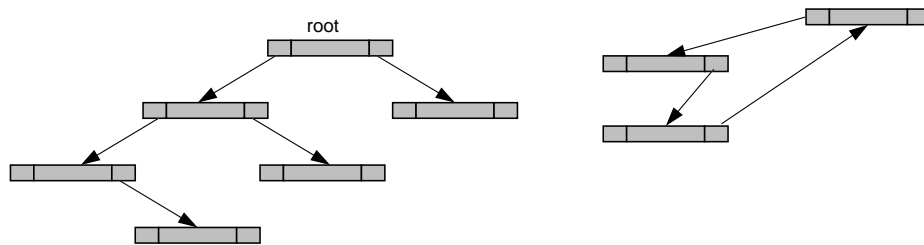
a.



b.



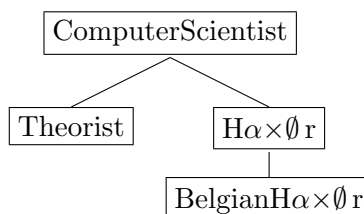
c.



Problem 6.3

This problem is to be done electronically. Start off by copying the stencil file from /course/cs031/pub/hw07/vtbl.s This will give you some support code that will test the work you'll be doing.

Consider the class hierarchy and class definitions below.



```

public abstract class ComputerScientist {
    public ComputerScientist(int num) { _favoriteNumber = num; }
    public int getFavoriteNumber() { return _favoriteNumber; }
    public abstract void tellJoke();
    public abstract void tellPunchline();
    private int _favoriteNumber;
}

public class Theorist extends ComputerScientist {
    public Theorist() { super(51); }
    public void tellJoke() {
        System.out.println("What did NP say to circuit-sat?");
    }
    public void tellPunchline() {
        System.out.println("\nYou complete me.\n Isn't that phat?");
    }
}

public class Haxor extends ComputerScientist {
    public Haxor() { super(31337); }
    public void tellJoke() {
        System.out.println("Why do computer scientists get Halloween and Christmas confused?");
    }
    public void tellPunchline() {
        System.out.println("Because oct 31 = dec 25.");
    }
    public void tellWhatChanges() {
        System.out.println("uhramm");
    }
    public void tellWhatChangesNot() {
        System.out.println("uhromm");
    }
}
  
```

```

}

public class BelgianHaxor extends Haxor {
    public BelgianHaxor() { super(); }
    public void tellWhatChanges() {
        System.out.println("uhromm");
    }
    public void tellWhatChangesNot() {
        System.out.println("uhromm");
    }
    public void tellJoke() {
        System.out.println("Who is following me?");
    }
    public void tellPunchline() {
        System.out.println("Oh, gosh. . .");
    }
}
}

```

40

The support code contains three main parts:

- A data section declaring important constants and strings and allocating space for an instance of one of each of the three concrete types.
- A main subroutine that will use your VTBLs to call methods on each type of object
- Subroutines that print out strings. You will reference these labels in your VTBLs.

Do the following:

- a. As a warm-up, write the `cs_get_favorite_number` subroutine in proper MIPS assembly.

It takes one parameter, the address of a `ComputerScientist` object, in `$a0`, and returns the `ComputerScientist`'s favorite number in `$v0`.

- b. For the rest of this problem, you will implement virtual function tables and related initialization subroutines.

Add code to the end of the data segment that defines the following three labels, providing enough storage space for each one.

```

__theorist_vtbl:
__haxor_vtbl:
__belgian_haxor_vtbl:

```

Note that you don't need a `__cs_vtbl`; you may want to think about why that is the case.

How many vtbls will you need? How many words of memory are needed to store each vtbl? (Are they all the same size?)

- c. Write the `vtbl_init` subroutine that populates your vtbls with appropriate function addresses.

This subroutine takes no parameters and returns nothing.

You will probably want to reference the following labels, which at this point should have already been fully defined.

```
cs_get_favorite_number
say_uhramm
say_uhromm
say_theorist_joke
say_theorist_punchline
say_haxor_joke
say_haxor_punchline
say_belgian_haxor_joke
say_belgian_haxor_punchline
```

- d. Write the `cs_init` subroutine that initializes a new instance of the `ComputerScientist` class.

It should take a pointer (in `$a0`) to 8 bytes of memory that have been pre-allocated for the new instance, and a "favorite number" (in `$a1`) and doesn't return anything.

- e. Write the remaining initialization subroutines, being sure that each initialization subroutine makes a call to the initialization subroutine of the parent class.

```
theorist_init
haxor_init
belgian_haxor_init
```

Each of these should take a pointer in `$a0` to the memory allocated for the new object.

We've already written code to test your VTBLs, assuming you implement them to match our specifications. Our code runs through each type of object (theorist, haxor, belgian haxor) and calls all of their methods. Thus, if you actually run your code, you should see:

```
51
What did NP say to the circuit-sat?
You complete me.
31337
Why do computer scientists get Halloween and Christmas confused?
Because oct 31 = dec 25
uhramm
uhromm
31337
Who is following me?
Oh gosh...
uhromm
uhromm
```

Hand in a single MIPS file incorporating your answers for parts a - f.