System Software Project

Memory Management

The goal of this project is to implement a heap with a freelist as well as a Mark & Sweep garbage collector. Since it is a large project you can implement it in teams of two students. It is recommended to use C++ or C as an implementation language because the low-level programming required for a garbage collector is hard (if not impossible) to do in Java or C#.

1. Memory Allocation

Implement a class `Heap` representing a heap of size 32 Kbytes. The list of free heap blocks should be managed with the *first-fit method*. Objects should be allocated with an `alloc` function. In order to allocate an object of a class `Student`, say, one should call

```
adr = Heap::alloc("Student");
```

This function should allocate an object with the required size, install in it a pointer to the type descriptor of class `Student`, and return the address of the new object.
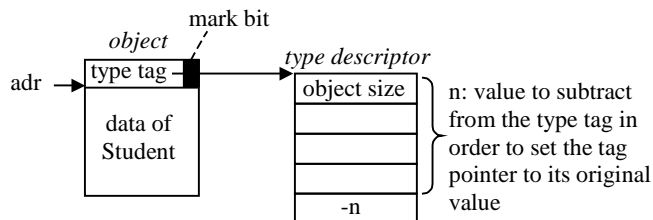
The least significant bit of the type tag is used as the mark bit of the garbage collector.

For every class (e.g. `Student`) there is exactly one type descriptor. Normally, type descriptors are generated by the compiler. In your project, however, they should be created and initialized by the user program. The type descriptor of every class should be registered at the heap using the call:

```
Heap::registerType("Student", studentDescAdr);
```

The heap maintains a list of all type descriptors and assigns them to the type tags of newly allocated objects.

In the beginning the free list contains a single large block (i.e., the whole heap).

2. Garbage Collector

Implement a Mark & Sweep garbage collector using the Deutsch-Schorr-Waite algorithm. It should be invoked by

```
Heap::gc(roots);
```

where roots is an array of root pointers terminated with a null value. The mark phase should traverse and mark all objects that can be directly or indirectly reached from the roots (use the least significant bit of the type tag for marking an object). You can assume that the method call stack does not contain any pointers when gc is called. The sweep phase should build a new freelist and merge adjacent free blocks.

For testing the garbage collector, implement a method

```
Heap::dump();
```

which prints a list of all live objects as well as a list of free blocks. For every live object the following values should be printed:

- address (hexadecimal)
- name of the object's type
- the first 4 bytes of the object in hex form (to get a clue of the object's contents)
- a list of all pointers in this object (in hexadecimal form)

dump() should also print the total amount of memory used by live objects.

When dumping the freelist the address and length of every free block should be printed as well as the total amount of free memory.

3. Test program

Test your memory management system with a real application. For example, you could implement a program that manages students and courses according to the following class diagram.
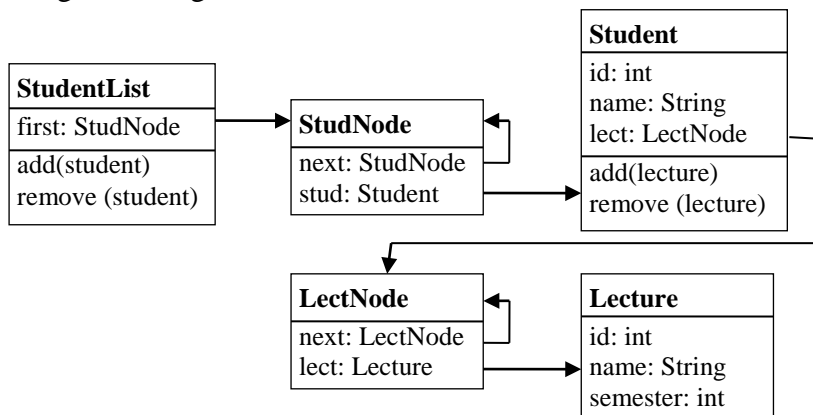
A snapshot of such a data structure could look as follows:
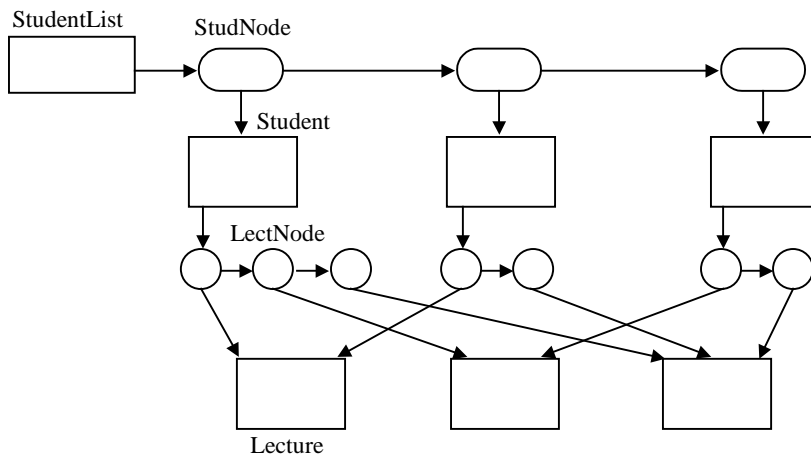
At the start of your program create all type descriptors for your classes and register them at the heap. In order to create a Student object in C++, you can then write

```
Student s = (Student) Heap::alloc("Student");
```

Build a sufficiently complex data structure with your objects and then delete some objects by removing all pointers to them. Call the garbage collector and check if all unreferenced objects are collected correctly. Finally delete the pointer to the StudentList node and call the garbage collector again. The heap should now consist of a single free block again.

Assume that your program has a single root pointer pointing to the StudentList node.