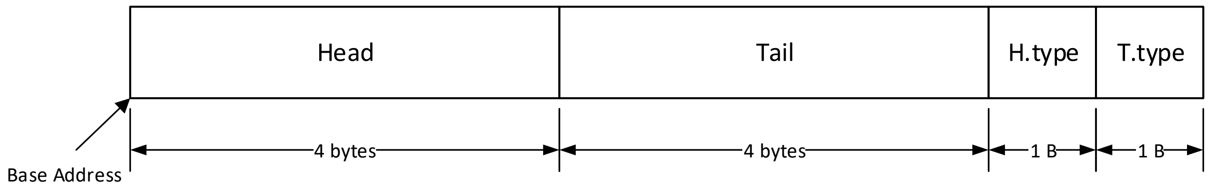
Scheme is one of the purer examples of a functional language. It attempts to eschew maintaining a state and encourages the programmer to use recursion to solve complex problems. The primary method of computation in Scheme involves the construction and manipulation of lists. The language provides three functions which allow the programmer to get the first piece of data from the list (car), remove the first element and retain the rest of the list (cdr) and concatenate two pieces of data or lists (cons).

For this assignment, you will be implementing the three list manipulation functions using a simple data structure. Once those are working, you will implement a recursive double function that uses the list manipulation functions to double any integer in the list. The data structure you should implement is similar to a linked list, since it can point to the next element in the list. However, it can also contain lists as the data, or contain data instead of having another element in the list. Since we will need to know what the data type for both elements is, we will want to have a datatype attached to both elements. We will only be interested in a few different datatypes (integer, character, list, null), so we will only need one byte of storage for the datatype, but will want four bytes for the element itself. Thus, the data structure requires 10 bytes of storage. Since we need to make the element word-aligned, we will place the data types at the end of the data structure.



Your functions should operate regardless of the data that they receive. In many cases they will simply return their parameters. If your function really cannot operate on the provided parameters, you should return an empty list. You should expect to receive a piece of data in $a0 and a corresponding datatype in $a1. Your function should return some data in $v0 and the associated datatype in $v1.

Since the test suite will contain lots of tests that you won't be prepared for initially, you should start by writing four dummy functions that simply return a default value (0 and 0 is a good choice) so that the program doesn't crash.

You will be provided with a series of helper functions that will tell you about a given data type. This will help you to determine when to perform recursion, and when to stop.

You should not need to modify the test suite. You only need to write the functions. Since QT SPIM expects all of your code to be in a single file, you can concatenate them together in a few ways. If you are on Windows, you can use the included batch file to do the work for you. Simply dragging your source file and dropping it on the batch file should be sufficient. If you are having trouble with the batch file, make sure that your file names match those below. You can also use a command line operation.

Windows: `copy /Y "<Your Source File Name>"+"Test Suite.asm" Output.asm`

Unix: `cat "<Your Source File Name>" "Test Suite.asm" > Output.asm`

Your program should include appropriate comments indicating what the code should be doing and what registers are being used for. After displaying the results, your program should exit cleanly. You should test your programs using the SPIM simulator to ensure their functionality before submitting them. You should only submit your four functions. You will not receive credit if you submit the test suite in any form. You should also not include any driver or debug code in your submission.

Objectives:

1. To introduce dynamic data structure creation.
2. To introduce data structure manipulation.
3. To review recursive algorithm construction.
4. To review stack manipulation.
5. To review parameter checking.