CMPT350: Lab Demo 8

Creating a struct with dynamic memory *1%*

Due: November 23rd, 2022 @ 17:00

Dynamically allocated strings are useful, but we can do much more with it. In this lab, we will work towards creating a linked list, dynamically creating one linked list node to hold a name and id number.

Laboratory Procedure:

On the course Moodle page, look for the Demo 8 assignment in the Laboratories section. Inside is a premade skeleton MIPS file, `demo8.s`. Save this somewhere handy on your machine. You may use both the command line version, `spim`, and the GUI version, QtSpim to run your program. As well, you will need a text editor to edit your programs.

Assignment Requirements:

- Prompt a user for a number (their id number)
- Prompt and store the user's name dynamically (what we did in demo 7)
- Dynamically allocate enough memory to hold one linked list node. The node holds
  - The value of the user's id
  - The address of the memory location which holds the user's name
  - The address of the next node in the linked list. Since this is the only node for this assignment, this value should be set to zero (a.k.a the "null" address)
- Store the id in the node in the first word of the node
- Store the address holding the name in the second word of the node
- Store the address of the next linked list node (0) in the third word of the node
- Print out the address of the node, the id, the name and the address of the name, and the address of the next node.

*** **NOTE:** Your code **must compile and execute** within the **laboratory environment**. Failure to do so will result in a **mark of ZERO** for the program. ***

Submission

Your programs should be named using your name and end in a .s extension. For example, BakerDemo8.s

When you feel your programs meet all of the above requirements, call the lab coordinator to demonstrate your program, and create a zip archive with your code submission and trace file(s) included. An easy way to do this is with the `zip` command. For example, if your solution file and trace file are in a folder called `baker`, then the command `zip -r baker.zip baker/` will create a zip file containing your submission files called baker.zip (Replace this with your name when creating your zip file). Upload this zip file to Moodle.

Sample Run

```
qbaker@S211-2-01: ~/350/demos/8
(v) qbaker@S211-2-01: ~/350/demos/8$ spim load demo8_solution.s
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Hello, lets create an id entry for you
Please enter your id: 12345
How many characters are in your name? 5
Please enter your name: Quinn
Node data for node 10020000
id: 12345
name/address: Quinn 1002000C
next node address: 00000000
(v) qbaker@S211-2-01: ~/350/demos/8$
(v) qbaker@S211-2-01: ~/350/demos/8$
```

Hints

- Your Demo 7 solution will be a great starting point, since we will be reading the name from the user the same way in this demo.
- To determine how much memory to allocate for the node, think about the contents of the node:
    - The id, an integer
    - The address of the name string, a memory address
    - The address of the next node, another memory address

  Adding the size of each of these tells us how many bytes we need to allocate

- A function is provided in the skeleton file to print an integer as a hexadecimal value. This is optional but recommended to print the memory address. Refer to the skeleton file for documentation about how the function works
- Remember that dynamic memory is still memory. We use our `sb/lb` and `sw/lw` instructions to write and read to memory that we allocate. For this lab, it is useful to think of the pieces of our node in terms of words, so `sw/lw` will be useful.
- A function to print our memory addresses as hexadecimal values has been provided in the skeleton. **Remember our function conventions**:
    - Use `jal` to call the function
    - Save your registers - **$t, $a, and $v registers may be overwritten by the function call**
- Refer to the function documentation for usage details (including arguments and return value)