CS330 Project #3

DUE: Wednesday, April 8, 11:59 PM

1 Objectives

- Implementation of a very simple interpreter
- Simulating a run-time heap

2 Problem Statement

You are to build a parser for a *very* simplified version of the grammar from project 2. This parser will essentially run a sequence of requested function calls corresponding to run-time heap management. The new grammar is specified below:

$$\begin{array}{lll}
\textit{program} & \rightarrow & \textit{function} \\
\textit{function} & \rightarrow & \textbf{LBRACK VOID RBRACK ID LPAREN RPAREN } \textit{compound} \\
\textit{compound} & \rightarrow & \textbf{BEGIN } \textit{stmtlist} \textbf{ END} \\
\textit{stmtlist} & \rightarrow & \textit{stmt} \mid \textit{stmt} \textbf{ SEMICOLON } \textit{stmtlist} \\
\textit{stmt} & \rightarrow & \textbf{ID} \mid \textbf{ID LPAREN NUM\_INT RPAREN} \mid \\
 & & \textbf{ID ASSIGNOP ID LPAREN NUM\_INT RPAREN} \mid
\end{array}$$

Essentially, a <*stmt*> can only be a function call. The function calls that can be made will be the following (note that they are case *sensitive*:

- **ID = Malloc(NUM_INT)**, which will allocate **NUM_INT** bytes from the heap and bind them the the variable specified by **ID**. This allocation should be done in a *best fit* manner and no implicit compaction should be performed. If the request will not fit in the heap, enough space should be added to the end of the heap to satisfy the request (i.e. the heap should be grown on an as needed basis).
- **Delete(ID)**, which simply restores the space allocated to the variable specified by **ID** to the heap. No implicit compaction should be performed.
- **PrintHeap**, which should simply print out the current free cells in the heap's freelist. Each free cell should be described by its beginning and ending adresses and its size.
- **PrintVars**, which should print out all of the currently bound variables and their allocations. Each variable should have its name, current address (at least begin, but you can also include end), and current size.
- **PrintBlank**, which should just print a blank line to the screen. *This is probably the easiest of all!*
- **PartialCompact**, which should partially compact the heap.
- **FullCompact**, which should perform a full compaction on the heap.

3 Implementation Details

- Write a main function that reads two command line arguments. The first is the source code file to process, and the second is to be the *initial* size of the heap (in bytes).
- from there, all you have to do is make sure to follow the above specifications!

4 What and How to Submit

Submit the following to the `classes.cs.siue.edu` website, or e-mail it to me (`sblythe@siue.edu`):

- all of your source code.
- a file called `read.me`, which describes your project. Partial credit abounds from information found in this file, so *do not treat this part lightly!*

5 Extra Credit

If you submit this project 48 hours or more early (i.e. 4/6 at 11:59 PM), you will get a free 5 point bonus.

6 Grading Breakdown

Successful compilation	30%
Correct execution	50%
Comments, Coding Style, & read.me file	20%
Extra Credit	5%

7 Final Notes

- START <u>NOW !</u>
- Do <u>NOT</u> try and enhance the given grammar. DO <u>NOT</u> WASTE TIME BY DOING UNNECESSARY WORK!
- *START <u>NOW !</u>*
- If you have *any* questions about *anything* regarding this project, SEE ME IMMEDIATELY !!!!!
- **START <u>NOW !</u>**

8 Example Run

In what follows, the percent (`%`) sign is the command prompt and `heapify` is the executable name on a Linux system:

```
% cat test.myl
[void] myfunc()
begin
  a=Malloc(10); b=Malloc(123); c=Malloc(1000);
  PrintVars; PrintHeap; PrintBlank;

  Delete(c); Delete(a); Delete(b);
  PrintVars; PrintHeap; PrintBlank;

  b=Malloc(8); c=Malloc(120);
  PrintVars; PrintHeap; PrintBlank;

  PartialCompact;
  PrintVars; PrintHeap; PrintBlank;

  PartialCompact;
  PrintVars; PrintHeap
end
```

(continued on next page ...)

```
% ./heapify test.myl 1024
Current list of dynamic variables with allocations:
a-->[0 ... 9] (10)
b-->[10 ... 132] (123)
c-->[133 ... 1132] (1000)
==== end of variable listing ====
Heap freelist is currently:
   <empty>
==== end of heap freelist ====

Current list of dynamic variables with allocations:
   <empty>
==== end of variable listing ====
Heap freelist is currently:
[10 ... 132] (123)
[0 ... 9] (10)
[133 ... 1132] (1000)
==== end of heap freelist ====

Current list of dynamic variables with allocations:
b-->[0 ... 7] (8)
c-->[10 ... 129] (120)
==== end of variable listing ====
Heap freelist is currently:
[130 ... 132] (3)
[8 ... 9] (2)
[133 ... 1132] (1000)
==== end of heap freelist ====

Current list of dynamic variables with allocations:
b-->[0 ... 7] (8)
c-->[10 ... 129] (120)
==== end of variable listing ====
Heap freelist is currently:
[130 ... 1132] (1003)
[8 ... 9] (2)
==== end of heap freelist ====

Current list of dynamic variables with allocations:
b-->[0 ... 7] (8)
c-->[8 ... 127] (120)
==== end of variable listing ====
Heap freelist is currently:
[128 ... 1132] (1005)
==== end of heap freelist ====
```