Linked Structures: Process Management Simulation

The purpose of this assignment is to demonstrate your ability to design and implement linked structures to store and manage data elements, and to implement data elements that have substantial dynamic content. This project requires blending and modifying the first and third projects to achieve a fairly interesting simulation of a process management system.

The system will use a doubly-linked list to store process control blocks (PCBs) similar to those from the first project, but adding a call stack. The underlying physical structure for the call stack must be a singly-linked list.

As before, the stack must be encapsulated within a class, and provide a general interface suitable to a stack. In addition, the doubly-linked list must also be encapsulated within a class, and must also provide a general interface suitable to a list. That is, neither the stack nor the linked list should make use of any specialized interface details of the objects that are stored within them.

The PCBs must also be implemented as a class, again with a suitable interface. Since the PCB will encapsulate the stack of activation records, the PCB will have to provide some interface elements to provide for managing function calls, local variable updates, and function returns. Of course, that means that the PCB will also use the activation record interface.

The activation record must also be implemented as a class. Ideally, you will be able to use the activation record class from the previous project without modification.

There is a significant organizational change for the process list; now the process priority becomes important. When a context switch is carried out, the command will specify the priority of the next process to be moved to the run state. Logically, you might think of it as if you have a collection of process lists, one for each possible priority. (To simplify that, we will guarantee that priorities will range from 0 to 4.) Thus, you might have an array of process queues, each reserved for a particular priority, or have a single process queue, which may (or may not) keep the PCBs in order of priority.

Script file:

This project will involve only one input file, containing the commands that are to be processed. The script file will contain a sequence of commands, one per line. Most of the commands are precisely the same as for projects one and three, but the specifications of all commands are repeated here for clarity. The two sort commands from project one have been eliminated; so have the display and clear commands from project three.

`start``<tab><process name><tab><priority><newline>`

Create a PCB for the process and append it to the relevant process queue. The system assigns a unique process identifier (PID); the PID is just a nonnegative integer, corresponding to the sequence in which the processes are created, starting with zero.

`kill``<tab><PID><newline>`

Search for a PCB storing the given PID. If one is found, remove the PCB. Note the PCB may be in the run state or on the process queue; deletion may leave either of those empty. (There's no automatic context switch.)

`switch``<tab><priority><newline>`

If there is no process with the specified priority in the process queue this has no effect. Otherwise, the first PCB with that priority in the process queue is moved to the run state and the running process is moved to the end of the relevant process queue.

`ps``<tab>[ `**`all`**` | PID ]<newline>`

If the parameter is `all`, log all the existing processes, showing the process name, PID, and current state (RUNNING or ELIGIBLE), and the process priority. The running process, if any, should be logged first, followed by the processes in the process queue, ranked in order of priority. If the parameter is a `PID`, locate the PCB for the specified process and log it as described above.

call<tab><fn name><tab><# of parameters><tab><values of parameters><tab>
<# of locals><tab><values of locals><newline>
Create an activation record for the function and push it onto the run-time stack. Log a message recording the initiation of the call.

return<tab><newline>
Pop an activation record off of the run-time stack. If stack is empty when this command is encountered, terminate the relevant process immediately. If the stack is empty immediately after this command is processed, terminate the relevant process immediately. Log a message recording the termination of the function, and the process if that occurs.

set<tab><position><tab><value><newline>
Set the value of the specified local variable to the given value. The position corresponds to the order in which the local variables are stored; they are numbered starting at one. If there is no such local variable, then log an error message, and terminate the process.

exit<tab><newline>
Immediately stop reading the script and exit the program.

Legend:

process name	
fn name	a character string, containing no tabs, with no more than 20 characters
priority	a non-negative integer value; higher values indicate greater priority
PID	a non-negative integer value identifying a process, assigned upon process creation
# of parameters	a non-negative integer value
# of locals	a non-negative integer value
values of parameters	a tab-separated list of the specified number of integer values
values of locals	a tab-separated list of the specified number of integer values
position	a positive integer value possibly identifying a local variable
value	an integer value

As a general rule, every command you process should result in a logged message, either confirming the command was carried out or indicating that an error occurred. There is no limit on the number of commands that may be given. Your program should be designed to stop when an input failure is reached. There is a sample script file at the end of this specification.

Log file:

Your program will write all of its output to a log file. The contents of the log file should be similar to the previous projects. However, because this project will not be auto-graded, you now have considerable leeway in formatting and setting text for the feedback messages. The earlier requirements for logging information still apply. There is a sample log file at the end of this specification.

You must echo the names of the script and log files at the beginning of the log. To aid in examining your log output, you must number the commands (starting at 1).

Program invocation:

Your program will normally be invoked from the command line environment. Assuming your executable file is called ProcessManager.exe, it would be invoked as follows:

```
ProcessManager   <script file name>   <log file name>
```

You should check for existence of the named script file, and terminate execution with an informative message if the file is not found.

Evaluation:

Your submitted program will be evaluated by one of the TAs, at a demo, which you will schedule. Details about scheduling the demos will be announced later. The TA will also be evaluating your implementation for correctness of operation, as well as for design quality and documentation.

Your implementation must meet the following requirements:

- Choose descriptive identifiers when you declare a variable or constant. Avoid choosing identifiers that are entirely lower-case.
- Use named constants instead of literal constants when the constant has logical significance.
- Use C++ streams for input and output, not C-style constructs.
- Use C++ `string` variables to hold character data, not C-style character pointers or arrays.
- In the procedural part of your implementation, you must make appropriate use of functions.
- None of your functions can include more than 30 executable statements. An executable statement is one that causes something to happen. Comments, constant and variable declarations (even if they initialize) do not count as executable statements.
- When you pass an array to one of your functions, use pass by constant reference unless pass by reference is logically necessary. (Remember that for array parameters, pass by reference is the default.)
- You must use dynamic allocation as described in this specification.
- The process control blocks and activation records must be implemented as classes.
- The stack must be implemented as a C++ class, and the underlying structure must be a singly-linked list. Templates are banned, including all STL containers.
- The process list(s) must be implemented as a class, and the underlying structure must be a doubly-linked list.
- Each class must be implemented using a header file containing the class declaration and an associated source file containing the implementations of the class functions.
- Any class that allocates memory dynamically must include a destructor to deallocate that memory, and a copy constructor and assignment operator to provide correct deep copy support. (This might include all of your classes.)

Read the *Programming Standards* page on the CS 1704 website for general guidelines. You should comment your code in some detail. In particular:

- You should have a header comment identifying yourself, and describing what the program does.
- Every constant and variable you declare should have a comment explaining its logical significance in the program.
- Every major block of code should have a comment describing its purpose.
- Every function implementation must have a header comment. The format of the header comment is described in the course notes, as well as on the *Programming Standards* page on the course website.
- Every class declaration must have a header comment, describing the purpose of the class and anything a client would need to know about the limitations of the class.
- Adopt a consistent indentation style and stick to it.

Understand that the list of requirements here is not a complete repetition of the *Programming Standards* page on the course website. It is possible that requirements listed there will be applied, even if they are not listed here.

Submitting your program:

You will submit this assignment, as a standard zipped archive, to the Curator System (read the *Student Guid*e). Do not use any other format for the archive file. However, this assignment will not be graded automatically. Follow the instructions on the website for submitting projects that are not auto-graded. Be sure to note that more than just your source files are required, and be sure to not submit any extra files.

You will be allowed up to three submissions for this assignment. Only one should be necessary, but this will allow you a chance to fix errors you may find after making your first submission.

The *Student Guide* and submission link can be found at: http://www.cs.vt.edu/curator/

Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the pledge statement provided on the course website in the header comment for your program.

Failure to include the pledge statement may result in a substantial grade penalty.

Sample input script:

```
; PCB List Script file
;
; Load some processes:
start    P01   0
ps 0
start    P02   3
start    P03   0
;
; Check the status of the jobs:
ps all
;
; Put a job into the run state:
switch   0
;
; Verify:
ps all
;
; Call some functions:
call     main  0     3     1     2     3
call     F01   1     2     2     3     5
;call    F02   3     1     2     3     0
ps 0
set      1     100
ps 0
;
; Do a context switch and verify:
switch   0
ps all
;
; Some calls:
call     main  0     1     1
call     G01   0     0
call     G02   1     3     2     1     2
ps 2
;
; Another context switch:
switch   0
;
; Try a function return:
return
ps 0
;
```

```
; And now terminate main():
return
ps 0
;
; And finally try a return with no running process:
return
;
; Verify process list:
ps all
;
; Add another process:
start    P04   2
ps 3
;
; Run another process:
switch   3
call     main  0     0
ps 1
;
; Try a switch w/o a candidate process to
; move to run state:
switch   3
ps all
;
; Kill one and verify:
kill     2
kill     0
ps all
;
switch   1
;
; Shut down:
exit
```

Sample log:

```
Programmer: Bill McQuain
CS 1704:    Process Management Simulation

Script file: 1.Script.txt
Log file:    1.Log.txt
--------------------------------------------------------------------------
Command   1: start   P01   0
Process 0 added.
--------------------------------------------------------------------------
Command   2: ps      0
P01            PID:    0     Priority:    0
--------------------------------------------------------------------------
Command   3: start   P02   3
Process 1 added.
--------------------------------------------------------------------------
Command   4: start   P03   0
Process 2 added.
--------------------------------------------------------------------------
Command   5: ps      all
Processes with priority 0:
P01            PID:    0     Priority:    0
P03            PID:    2     Priority:    0

Processes with priority 1:

Processes with priority 2:

Processes with priority 3:
P02            PID:    1     Priority:    3

Processes with priority 4:

--------------------------------------------------------------------------
Command   6: switch  0
Process 0 now in run mode.
--------------------------------------------------------------------------
Command   7: ps      all
Running process:
P01            PID:    0     Priority:    0
Call stack:
     Stack is empty.


Processes with priority 0:
P03            PID:    2     Priority:    0

Processes with priority 1:

Processes with priority 2:

Processes with priority 3:
P02            PID:    1     Priority:    3

Processes with priority 4:
```

```
------------------------------------------------------------------------
Command   8: call    main  0     3     1     2     3
Call to main initiated.
------------------------------------------------------------------------
Command   9: call    F01   1     2     2     3     5
Call to F01 initiated.
------------------------------------------------------------------------
Command  10: ps      0
P01            PID:    0     Priority:    0
Call stack:
     Function:   F01
     Parameters:        2
     Locals:            3       5

     Function:   main
     Parameters:
     Locals:            1       2       3


------------------------------------------------------------------------
Command  11: set     1     100
------------------------------------------------------------------------
Command  12: ps      0
P01            PID:    0     Priority:    0
Call stack:
     Function:   F01
     Parameters:        2
     Locals:          100       5

     Function:   main
     Parameters:
     Locals:            1       2       3


------------------------------------------------------------------------
Command  13: switch  0
Process 2 now in run mode.
------------------------------------------------------------------------
Command  14: ps      all
Running process:
P03            PID:    2     Priority:    0
Call stack:
     Stack is empty.


Processes with priority 0:
P01            PID:    0     Priority:    0

Processes with priority 1:

Processes with priority 2:

Processes with priority 3:
P02            PID:    1     Priority:    3

Processes with priority 4:

------------------------------------------------------------------------
```

```
Command  15: call    main  0     1     1
Call to main initiated.
--------------------------------------------------------------------------
Command  16: call    G01   0     0
Call to G01 initiated.
--------------------------------------------------------------------------
Command  17: call    G02   1     3     2     1     2
Call to G02 initiated.
--------------------------------------------------------------------------
Command  18: ps      2
P03            PID:    2     Priority:    0
Call stack:
     Function:   G02
     Parameters:        3
     Locals:            1       2

     Function:   G01
     Parameters:
     Locals:

     Function:   main
     Parameters:
     Locals:            1


--------------------------------------------------------------------------
Command  19: switch  0
Process 0 now in run mode.
--------------------------------------------------------------------------
Command  20: return
Function F01 terminated for process P01
--------------------------------------------------------------------------
Command  21: ps      0
P01            PID:    0     Priority:    0
Call stack:
     Function:   main
     Parameters:
     Locals:            1       2       3


--------------------------------------------------------------------------
Command  22: return
main terminated for process P01
--------------------------------------------------------------------------
Command  23: ps      0
Process not found.
--------------------------------------------------------------------------
Command  24: return
There is no running process.
--------------------------------------------------------------------------
```

```
Command  25: ps      all
Processes with priority 0:
P03            PID:    2     Priority:    0

Processes with priority 1:

Processes with priority 2:

Processes with priority 3:
P02            PID:    1     Priority:    3

Processes with priority 4:

------------------------------------------------------------------------
Command  26: start   P04   2
Process 3 added.
------------------------------------------------------------------------
Command  27: ps      3
P04            PID:    3     Priority:    2
------------------------------------------------------------------------
Command  28: switch  3
Process 1 now in run mode.
------------------------------------------------------------------------
Command  29: call    main  0     0
Call to main initiated.
------------------------------------------------------------------------
Command  30: ps      1
P02            PID:    1     Priority:    3
Call stack:
     Function:   main
     Parameters:
     Locals:


------------------------------------------------------------------------
Command  31: switch  3
No eligible processes.
------------------------------------------------------------------------
Command  32: ps      all
Running process:
P02            PID:    1     Priority:    3
Call stack:
     Function:   main
     Parameters:
     Locals:



Processes with priority 0:
P03            PID:    2     Priority:    0

Processes with priority 1:

Processes with priority 2:
P04            PID:    3     Priority:    2

Processes with priority 3:
```

```
Processes with priority 4:

--------------------------------------------------------------------------
Command  33: kill    2
Process 2 removed.
--------------------------------------------------------------------------
Command  34: kill    0
Process not found: 0
--------------------------------------------------------------------------
Command  35: ps      all
Running process:
P02            PID:    1     Priority:    3
Call stack:
     Function:   main
     Parameters:
     Locals:



Processes with priority 0:

Processes with priority 1:

Processes with priority 2:
P04            PID:    3     Priority:    2

Processes with priority 3:

Processes with priority 4:

--------------------------------------------------------------------------
Command  36: switch  1
No eligible processes.
--------------------------------------------------------------------------
Command  37: exit
Exiting script execution.
--------------------------------------------------------------------------
```