Project 3 CS 2413 Data Structures Due 11:59 PM March 20, 2017

Objectives: You will learn to construct and manipulate the linked list class for main memory management.

1. [20 pts] Implement the Linked List data structure along with constructors, copy constructor, overloaded = operator, ostream operator, add method, remove method, and destructor.
2. [20 pts] Implement the Process class along with constructors, copy constructor, overloaded = operator, overloaded == operator (equal-to), ostream operator, and destructor.
3. [50 pts] Design and implement the program *per the project description*. 50 pts are distributed as follows:
   a. [10 pts] Allocate memory for process as per the description; this will include implementing the find method to determine if the process is in the linked list.
   b. [10 pts] Remove a process from memory and de-allocate memory as per description.
   c. [10 pts] Print for each process the starting and ending address as per project description.
   d. [20 pts] Method to perform memory compaction.
4. [10 pts] Document your project thoroughly as the examples in the textbook. This includes but is not limited to header comments for all classes/methods, explanatory comments for each section of code, meaningful variable and method names, and consistent indentation.

Description: The main memory management is described in your textbook. The size of the main memory for your project is 128KB. Processes request contiguous space in memory in terms of KB units. Initially, the main memory is empty (is not allocated to any processes). When a process makes a request for memory, contiguous space that is large enough to hold the process is allocated to it. For example, if a process requests 10KB of memory and 12KB of memory is available, 10KB of space is allocated to the process and 2KB free space partition is created. It is possible for memory to contain several free spaces but they may not be contiguous. **Compaction** is the process of merging the free spaces into one contiguous block (see exercise problem at the end of Chapter 4). If a contiguous space is not available, but the amount of free space adds up to the requested space by the process a compaction operation is performed. If there space is not available, then the request is rejected and a message is printed.

If the process is allocated 10KB and requests an additional 3KB, we need to find free space of size 13KB or more. If such a free space is available the process is moved to this large space – the process will get new starting and ending address and the original space of 10KB is set to be free.

You can use the linked list class and the partition class defined in your textbook.

Data Structures: You are to build your own linked list data structure (you can copy code from your book if you choose). The Linked List class that you build should be templated and you will use the class Process as a template. The class Process has several fields (read Chapter 4 for details). Make sure that you implement all the standard methods for the Process class.

Input/Process/Output: The input to your program is described below. The number of lines of input is unknown.

```
1 10    // Process 1 is requesting 10KB of contiguous memory.
2 20    // Process 2 is requesting 20KB of contiguous memory.
```

3 3 // Process 3 is requesting 4KB of contiguous memory.
0 0 // 0 indicates printing. Print the contents of the memory.
-1 0 // Print the total amount of free space available.
1 2 // Process 1 is requesting an additional 2KB of memory. Needs 12KB of contiguous memory.
2 –4 // Process 2 is deallocating 4KB of main memory. It has a total of 16KB contiguous memory.
1 –12 // Process 1 has completed its executing and it is removed from the main memory.
2 0 // Remove process 2 from memory and free up all its allocated memory

Each process requests memory that can be either a positive number or a negative number as given above. Assume process *p* requests a *+x* amount of contiguous memory for the first time. It is allocated *+x* amount of contiguous memory if it is available (after a compaction, if needed). If the space is not available, then its request is rejected. If process *p* had been already been allocated memory, then *+x* indicates that it requires additional memory and its request is either granted or rejected based on availability. If a process that is currently allocated *y* amount of memory requests *–x* amount of memory, then the process space is adjusted to contain *y-x* amount of memory and *x* amount of free space is created.

After processing each request, output information on the space allocated to the process (for example, the begin address and end address). If memory is not allocated, then print out a statement to this effect. For a print command, print the contents of each node in the linked list.

An input file will be provided. Good luck!