# Lab 3 CS1: Stack and Queues Applications (total 100pts)

In this lab, you are provided with Stack, Queue, and PriorityQueue libraries that you will need to complete some application files. Your submission will include 4 applications programs (1 for each data structure, and 1 with a combination of all).

Grading:

- Each program is 25 pts
    - Header and comments (5 pts): write a header explaining what the program does and who worked on it. Also include the class and semester information.
    - Data input as specified (5 pts): each program has a different way of data entry, write as asked in order to obtain full credit here.
    - Data structure usage (5 pts): calling the correct functions of the correct data structure.
    - Memory management (5 pts): Don't forget to allocate correctly and free all pointers.
    - Correctness and screenshot (5 pts): Submit a code that does what it is supposed to, along with a screenshot of it running at your computer.

## I. Postfix Notation (Stacks … input expression using argv)

The postfix notation is when an equation is written with operators first, then its operands. Read a postfix expression into main using the argv parameter. Then, write a function that solves the expression and prints the result onto the screen. You can assume that the expression will only contain integers and the following operators: +, -, *, /, %, ^. The answer must be given as an integer. A detailed explanation of the notation can be found in the following Wikipedia article:

http://en.wikipedia.org/wiki/Reverse_Polish_notation

Examples from command line:

./postfix 3 4 +

7

./postfix 2 4 + 3 / 6 +

8

./postfix 2 4 ^ 2 * 5 % 2 –

0

## II. RPG Game (Queues … input #fighters, #healers, #enemies and their stats inside a file using argv to pass the filename to main)

In a turn based RPG (Role-Playing Game) game, battles are handled on a turn basis. The party member (ally) that attacks first depends on how fast the character (member) is. Using Queues, simulate a battle between fighters, healers, and enemies. The following struct can be used to represent a character:

```
typedef enum SpriteType{
        FIGHTER,
        HEALER,
        ENEMY
} SType;
typedef struct _GameSprite{
        SType type;
        uint id;                //a unique id to identify the character (normally this could be a string
                                holding the name)
        Queue* actions;         //store Boolean (false = do nothing, true = attack)
        uint speed;             //speed is a value between [0, 10] …
                                //10 - speed is number of false before a true
        uint hp;                //life points of the character (between 50 and 200 for allies, 1000 for
                                //the enemy)
        uint strength;          //for healer is how much hp it can restore to target
                                //for fighters and enemy is how much hp to take from target
        uint accuracy;          //how probable is to hit the target between (0, 100)
                                //bosses should have high attack but lower accuracy than fighters
                                //healers normally will have high accuracy in healing
} GameSprite;
```

A full RPG game would affect the stats of the characters using spells or items, but this should give you an idea of how to build a battle sequence. Now the question that remains is: Who will win? Each time that a character attacks, print the character and its hp, target and its hp, and outcome (hit or miss). When the program starts place the initial content of each GameSprite (hold inside an array) and create a while loop that runs while at least there is one member from either side (neither allies or enemies have all perished). During each iteration evaluate each Queue and perform an action if required, once the action is performed reset the action Queue (#false before a true and the true). **IMPORTANT: this program relies on random numbers, the seed for srand will be passed to main using argv, once read call srand with the argv[i] parameter converted into an int (atoi in stdlib.h).**

You can use the following actionProcess function:

```
//COMPLETE THIS FUNCTION
//this function is called when a true is found on the gs->actions' front
//each iteration the front is dequeued and enqueued back into the gs->actions structure
//returns true if action hit
//returns false if action missed
bool actionProcess(GameSprite* gs, ArrayList* players, ArrayList* Enemies){
	//random number between (0 and 100)
	//gs->accuracy within the same interval, so higher accuracy characters have higher chance of
	//hitting
	if(rand % 100< gs->accuracy){
		switch(gs->type){
			case ENEMY:
				//get a random ally and reduce its hp
				//ally->hp = ally->hp  - gs->strength;
				break;
			case FIGHTER:
				//get a random enemy and reduce its hp
				//enemy->hp = enemy->hp - gs->strength;
				break;
			case HEALER:
				//get a random ally (preferably that it does not have all hp) and heal it
				//could choose itself
				//(not required, but a PriorityQueue sorted by hp could help here)
				// ally->hp = ally->hp  + gs->strength;
				break;
		}
		return true;
	}
	return false;
}

//OTHER FUNCTIONS TO IMPLEMENT
void resetActionQueue(GameSprite* gs);
//first need generate ArrayList* allies and ArrayList* enemies
//the configuration is a file description that can be read to generate characters, fclose the file once read
//allies contain fighters and healers, enemies fight alone (no healers, and sometimes is just only 1)
//this function runs while checkOutcome returns 0
void runBattle(FILE* configuration);
//returns 0 if no winner (there are allies alive and enemies alive)
//          -1 if allies lost (all allies perished)
//           1 if allies won (all enemies perished)
int checkOutcome(ArrayList* allies, ArrayList* enemies);
```

Format of input file:

- Each line contains the definition of a character
- First we have the id of the character, so we can keep track of the players
- The first char after the id is holding the type of character it represents (F for fighter, H for healer, and E for enemy)
- Following, the speed stat. This number ranges from 0 to 10 both included, and when subtracting 10 minus that number yields the number of false before the true in the queue. The true states when the character can attack (or heal if character is a healer)
- Then we get the hp for the character
- We then pass the strength, which is how much life to recover when performing action (healer) or take from target (fighter and enemy)
- Finally, the accuracy comes as last parameter

Example input file (input.data):

250 F 4 200 70 60

180 H 5 110 20 70

200 F 7 105 50 95

10 E 3 1000 80 40

./game input.data 200  #may not reflect the the actual game

200 attacks 10

Hit! 200 has  105 hp and 10 has 950 hp

250 attacks 10

Hit! 250 has 200 hp and 10 has 880 hp

10 attacks 180

Miss! 10 has 880 hp and 180 has 110 hp

…

Battle Won in 14 turns!

## III. Process Scheduling (PriotityQueue … input Process list from File, pass filename to main using argv)

Operating Systems (OS) in computers are in charge of many tasks, process scheduling being one of them. Process Scheduling involves finding out what is the next process (running program) to be run by the CPU, whenever a Context Switch occurs. The OS can base its decision on multiple criteria, example: wait time, # of pending instructions, time to finish, random.

In this question, you are to write a code that receives a list of process as shown and prints how they are selected by the OS:

```
typedef struct _Process{
        uint pid;                       //a unique id for each process
        uint remainingInstructions;     //number of remaining instructions
        uint runTime;                   //amount nof time which the process has run
} Process;
```

Your main will contain a PriorityQueue with different operation modes based on the command line parameters. Such mode of operation may be: 1) the number of instructions remaining (take the maximum), 2) the amount of run time (take the minimum), and 3) random. Each time you take an element from the PriorityQueue you must modify its attributes so that its priority changes (decrement instructions by 1, increment runtime by 1). Once instructions reach 0 the process finished exection and t is not reinserted into the PriorityQueue. Provide the output of all process scheduling schemes. At the end also provide the average context switch among all processes, defined as the sum(required_switches * instructions) / total_instructions. Use the following ProcessPrinter:

```
void processPrinter(Object p){
        Process* pointer = (Process*)p;
        Process proc = *pointer;
        printf("(PID=%d, Instr = %d, RunTime = %d)", proc.pid, proc.remainingInstructions,
proc.runTime);
}
```

Format of input file:

- Main receives **3 parameters**, 1) the filename of the file that contains the process information, 2) the mode of operation (1 instructions, 2 runTime, or 3 random), and **3) the random seed initializer (only needed for mode 3)**
- First line contains the number of processes in the PriorityQueue
- Any following line contains two numbers, the PID and the number of instructions
- runTime is initialized to 0 when the program starts
- HINT: the PriorityQueue requires passing a comparator function. Write your three functions and pass the proper function pointer depending on the mode of operation.

Example input file (input.data):

```
3
122 6
421 2
293 3

./scheduler input.data 1
{(PID = 122, Instr = 6, RunTime = 0), (PID = 293, Instr = 3, RunTime = 0), (PID = 421, Instr = 2 , RunTime = 0)}
122
{(PID = 122, Instr = 5, RunTime = 1), (PID = 293, Instr = 3, RunTime = 0), (PID = 421, Instr = 2 , RunTime = 0)}
122
{(PID = 122, Instr = 4, RunTime = 2), (PID = 293, Instr = 3, RunTime = 0), (PID = 421, Instr = 2 , RunTime = 0)}
122
{(PID = 293, Instr = 3, RunTime = 0), (PID = 122, Instr = 3, RunTime = 3), (PID = 421, Instr = 2 , RunTime = 0)}
293
{(PID = 122, Instr = 3, RunTime = 3), (PID = 421, Instr = 2 , RunTime = 0), (PID = 293, Instr = 2, RunTime = 1)}
122
{ (PID = 421, Instr = 2 , RunTime = 0), (PID = 293, Instr = 2, RunTime = 1), (PID = 122, Instr = 2, RunTime = 4)}
421
{ (PID = 293, Instr = 2, RunTime = 1), (PID = 122, Instr = 2, RunTime = 4), (PID = 421, Instr = 1 , RunTime = 1)}
293
{(PID = 122, Instr = 2, RunTime = 4), (PID = 421, Instr = 1 , RunTime = 1), (PID = 293, Instr = 1, RunTime = 2)}
122
{ (PID = 421, Instr = 1, RunTime = 1), (PID = 293, Instr = 1, RunTime = 2), (PID = 122, Instr = 1, RunTime = 5)}
421
(PID = 421, Instr = 0, RunTime = 2) completed in 9 context switches
{(PID = 293, Instr = 1, RunTime = 2), (PID = 122, Instr = 1, RunTime = 5)}
293
(PID = 293, Instr = 0, RunTime = 3) completed in 10 context switches
{(PID = 122, Instr = 1, RunTime = 5)}
122
(PID = 122, Instr = 0, RunTime = 6) completed in 11 context switches
{}
Done scheduling processes!
Average Wait Time: 11 context switches
```

./scheduler input.data 2
{(PID = 122, Instr = 6, RunTime = 0), (PID = 421, Instr = 2 , RunTime = 0), (PID = 293, Instr = 3, RunTime = 0)}
122
{(PID = 421, Instr = 2 , RunTime = 0), (PID = 293, Instr = 3, RunTime = 0), (PID = 122, Instr = 5, RunTime = 1)}
421
{(PID = 293, Instr = 3, RunTime = 0), (PID = 122, Instr = 5, RunTime = 1), (PID = 421, Instr = 1 , RunTime = 1)}
293
{(PID = 122, Instr = 5, RunTime = 1), (PID = 421, Instr = 1 , RunTime = 1), (PID = 293, Instr = 2, RunTime = 1)}
122
{(PID = 421, Instr = 1 , RunTime = 1), (PID = 293, Instr = 2, RunTime = 1), (PID = 122, Instr = 4, RunTime = 2)}
421
(PID = 421, Instr = 0, RunTime = 2) completed in 5 context switches
{(PID = 293, Instr = 2, RunTime = 1), (PID = 122, Instr = 4, RunTime = 2)}
293
{(PID = 122, Instr = 4, RunTime = 2), (PID = 293, Instr = 1, RunTime = 2)}
122
{(PID = 293, Instr = 1, RunTime = 2), (PID = 122, Instr = 3, RunTime =3)}
293
(PID = 293, Instr = 0, RunTime = 3) completed in 8 context switches
{ (PID = 122, Instr = 3, RunTime =3)}
122
{ (PID = 122, Instr = 2, RunTime =4)}
122
{ (PID = 122, Instr = 1, RunTime =5)}
122
(PID = 122, Instr = 0, RunTime = 6) completed in 11 context switches
{}
Done scheduling processes!
Average Wait Time: 10 context switches

## IV. Last Time of Palindrome (Promise … pass palindrome into main using argv)

You might be asking yourselves if there is any particular application to palindromes, other than an exam question. Palindromes can actually be seen in Music, Biology, Optics, among others. As the last time you check for palindromes, you **must** use **one Queue and one Stack only** in addition to the string parameter. In particular, each data structure will hold characters, as always discarding the spaces and special characters. For convenience, the following function is provided:

```
/**
This function returns '' when the character is not a letter, and the uppercase letter when finding a letter
@param c the character to check
@return the uppercase version of param c or '' if an illegal character is found
*/
char getPalindromeChar(char c){
	//c is a copy of the passed parameter
	if(c >= 'a' && c <= 'z'){
		c = c - 32;
	}
	if(c >= 'A' && c <= 'Z'){
		return c;
	}
	return '';
}
```

Program structure:

```
//palinromeChecker.c
bool isPalindrome(char* str, uint length);
int main(int argc, char** argv){
	if(isPalindrome(argv[1], strlen(argv[1]))){
		printf("%s is a palindrome\n", argv[1]);
	}
	else{
		printf("%s is not a palindrome\n", argv[1]);
	}
	return 0;
}
bool isPalindrome(char* str, uint length){
	//TODO
	//insert your check, you are only allowed to use a Stack, a Queue, and the char
	//getPalindromeChar(char) helper function
}
```

Example from the command line:
./palindrome "racecar"
racecar is a palinrome

./palindrome “apple”
apple is not a palindrome

./palindrome “Wonton, not now!”
Wonton, not now! is a palindrome