Farhan Zaman
CS21200

10/30/18
10:00AM

HW 4 Report

This homework was intended to get us familiar and used to data structures such as stack, queues and priority queues and their applications to real world problems. They can be implemented using static array, dynamic array, or a linked list, and each of their implementations have their own basic functions and private member variables. We got to explore new data structures, stuff like n-queens, balanced parentheses, traffic how queues work in a day to day basis.

For question 1, here is my header file:

```
#ifndef __ZAMAN_FARHAN_HW4_Q1_H__
#define __ZAMAN_FARHAN_HW4_Q1_H__

#include <iostream>
#include <cassert>
using namespace std;

template <class Item>
class Stack
{
	private:
		Item* data;
		size_t used;
		size_t capacity;
	public:
		Stack(size_t initSize=30);
		~Stack();
		void push(const Item& entry);
		void pop();
		bool empty() const {return(used==0);}
		size_t size() const {return used;}
		Item top() const;
		void print();
		void swap(Stack<Item>& s2);
};

#include "Zaman_Farhan_HW4_Q1.cpp"
#endif
```

For question 1, here is my cpp file:

```
#ifndef __ZAMAN_FARHAN_HW4_Q1_CPP__
#define __ZAMAN_FARHAN_HW4_Q1_CPP__

#include "Zaman_Farhan_HW4_Q1.h"

template <class Item>
Stack<Item>::Stack(size_t initSize)
{
	data=new Item[initSize];
	capacity=initSize;
	used=0;
}

template <class Item>
Stack<Item>::~Stack()
{
	for(int i=0; i<used; i++)
	{
		pop();
	}
}

template <class Item>
void Stack<Item>::push(const Item& entry)
{
	assert(size()<capacity);
	data[used]=entry;
	used++;
}

template <class Item>
void Stack<Item>::pop()
{
	assert(!empty());
	used--;

}

template <class Item>
Item Stack<Item>::top() const
{
	assert(!empty());
	return data[used-1];
}
```

```
template <class Item>
void Stack<Item>::print()
{
	assert(!empty());
	for(int i=0; i<used; i++)
	{
		cout<<data[i]<<endl;
	}
}

template <class Item>
void Stack<Item>::swap(Stack<Item>& s2)
{
	if(this==&s2)
	{
		return;
	}

	Stack<Item> s3;
	for(int i=0; i<s2.size(); i++)
	{
		s3.data[i]=s2.data[i];
	}
	s3.used=s2.used;
	s3.capacity=s2.capacity;

	for(int i=0; i<used; i++)
	{
		s2.data[i]=data[i];
	}
	s2.used=used;
	s2.capacity=capacity;
	for(int i=0; i<s3.used; i++)
	{
		data[i]=s3.data[i];
	}
	used=s3.used;
	capacity=s3.capacity;
}

#endif
```

For question 2, here is my header file:

```
#ifndef __ZAMAN_FARHAN_HW4_Q2_H__
#define __ZAMAN_FARHAN_HW4_Q2_H__
```

```
#include <iostream>
#include <cassert>
using namespace std;

template<class Item>
class Queue
{
	private:
		Item* data;
		size_t first;
		size_t last;
		size_t count;
		size_t capacity;
		size_t nextIndex(size_t i) const {return(i+1)%capacity;}
	public:
		Queue(size_t initsize=30);
		~Queue();
		void push(const Item& entry);
		void pop();
		bool empty() const{return(count==0);}
		size_t size() const{return count;}
		Item front()const;
		void print();
		void swap(Queue<Item>& q2);
};

#include "Zaman_Farhan_HW4_Q2.cpp"
#endif
```

For question 2, cpp file:

```
#ifndef __ZAMAN_FARHAN_HW4_Q2_CPP__
#define __ZAMAN_FARHAN_HW4_Q2_CPP__

#include "Zaman_Farhan_HW4_Q2.h"

template <class Item>
Queue<Item>::Queue(size_t initsize)
{
	data=new Item[initsize];
	first=0;
	last=initsize-1;
	count=0;
	capacity=initsize;
}
```

```
template <class Item>
Queue<Item>::~Queue()
{
	for(int i=0; i<count; i++)
	{
		pop();
	}
}

template <class Item>
void Queue<Item>::push(const Item& entry)
{
	assert(size()<capacity);
	last=nextIndex(last);
	data[last]=entry;
	count++;
}

template <class Item>
void Queue<Item>::pop()
{
	assert(!empty());
	first=nextIndex(first);
	count--;
}

template <class Item>
Item Queue<Item>::front()const
{
	assert(!empty());
	return data[first];
}

template <class Item>
void Queue<Item>::print()
{
	assert(!empty());
	for(int i=first; i<last+1; i++)
	{
		cout<<data[i]<<endl;
	}
}

template <class Item>
void Queue<Item>::swap(Queue<Item>& q2)
{
	if(this==&q2)
	{
```

```
			return;
		}

		Queue<Item> q3;
		for(int i=q2.first; i<q2.last+1; i++)
		{
			q3.data[i]=q2.data[i];
		}
		q3.count=q2.count;
		q3.capacity=q2.capacity;
		q3.first=q2.first;
		q3.last=q2.last;

		for(int i=first; i<last+1; i++)
		{
			q2.data[i]=data[i];
		}
		q2.count=count;
		q2.capacity=capacity;
		q2.first=first;
		q2.last=last;

		for(int i=q3.first; i<q3.last+1; i++)
		{
			data[i]=q3.data[i];
		}
		count=q3.count;
		capacity=q3.capacity;
		first=q3.first;
		last=q3.last;
}

#endif
```

For question 3, here is my header file:

```
#ifndef __ZAMAN_FARHAN_HW4_Q3_H__
#define __ZAMAN_FARHAN_HW4_Q3_H__

#include <iostream>
#include <cassert>
using namespace std;

template <class Item>
class PriorityQueue
{
```

```
	private:
		Item* data;
		int* priority;
		size_t first;
		size_t last;
		size_t count;
		size_t capacity;
		size_t nextIndex(size_t i) const {return(i+1)%capacity;}
	public:
		void sort();
		PriorityQueue(size_t initsize=30);
		~PriorityQueue();
		void push(const Item& entry,int pr=0);
		void pop();
		bool empty() const{return(count==0);}
		size_t size() const{return count;}
		Item top() const;
		void print();

};

#include "Zaman_Farhan_HW4_Q3.cpp"
#endif
```

For question 3, here is my cpp file:

```
#ifndef __ZAMAN_FARHAN_HW4_Q3_CPP__
#define __ZAMAN_FARHAN_HW4_Q3_CPP__

#include "Zaman_Farhan_HW4_Q3.h"

template<class Item>
void PriorityQueue<Item>::sort()
{
	for(int i=last; i>0; i--)
	{
		if(priority[i]>priority[i-1])
		{
			int temp=priority[i-1];
			priority[i-1]=priority[i];
			priority[i]=temp;
			Item temp2=data[i-1];
			data[i-1]=data[i];
			data[i]=temp2;
		}
	}
```

```
}

template<class Item>
PriorityQueue<Item>::PriorityQueue(size_t initsize)
{
	data=new Item[initsize];
	first=0;
	last=initsize-1;
	count=0;
	capacity=initsize;
	priority=new int[initsize];
}

template<class Item>
PriorityQueue<Item>::~PriorityQueue()
{
	for(int i=0; i<count; i++)
	{
		pop();
	}
}

template<class Item>
void PriorityQueue<Item>::push(const Item& entry, int pr)
{
	assert(size()<capacity);
	last=nextIndex(last);
	data[last]=entry;
	priority[last]=pr;
	count++;
	if(pr>0)
	{
		sort();
	}
}

template<class Item>
void PriorityQueue<Item>::pop()
{
	assert(!empty());
	first=nextIndex(first);
	count--;
}

template<class Item>
Item PriorityQueue<Item>::top() const
{
	assert(!empty());
```

```
	return data[first];
}

template<class Item>
void PriorityQueue<Item>::print()
{
	assert(!empty());
	for(int i=first; i<last+1; i++)
	{
		cout<<data[i]<<endl;
	}
}

#endif
```

For question 4, here is my header file:

```
#ifndef __ZAMAN_FARHAN_HW4_Q4_H__
#define __ZAMAN_FARHAN_HW4_Q4_H__

#include "Zaman_Farhan_HW4_Q1.h"

template<class Item>
class Queue2
{
	private:
		Stack<Item> input;
		Stack<Item> reverse;
		size_t count;
		size_t cap;
	public:
		Queue2(size_t initsize=30);
		~Queue2();
		void push(const Item& entry);
		void pop();
		bool empty() const {return(count==0);}
		size_t size() const {return count;}
		Item front();
		void print();
};

#include "Zaman_Farhan_HW4_Q4.cpp"
#endif
```

For question 4, here is my cpp file:

```
#ifndef __ZAMAN_FARHAN_HW4_Q4_CPP__
#define __ZAMAN_FARHAN_HW4_Q4_CPP__

#include "Zaman_Farhan_HW4_Q4.h"

template<class Item>
Queue2<Item>::Queue2(size_t initsize)
{
	cap=initsize;
	count=0;
}

template<class Item>
Queue2<Item>::~Queue2()
{
	for(int i=0; i<count; i++)
	{
		input.pop();
	}
}

template<class Item>
void Queue2<Item>::push(const Item& entry)
{
	input.push(entry);
	count++;
}

template<class Item>
void Queue2<Item>::pop()
{
	while(reverse.empty()==false)
	{
		reverse.pop();
	}
	for(int i=count; i>0; i--)
	{
		reverse.push(input.top());
		input.pop();
	}
	reverse.pop();
	count--;
	for(int i=count; i>0; i--)
	{
		input.push(reverse.top());
		reverse.pop();
	}
}
```

```
template<class Item>
Item Queue2<Item>::front()
{
	while(reverse.empty()==false)
	{
		reverse.pop();
	}
	for(int i=count; i>0; i--)
	{
		reverse.push(input.top());
		input.pop();
	}
	Item temp=reverse.top();
	for(int i=count; i>0; i--)
	{
		input.push(reverse.top());
		reverse.pop();
	}
	return temp;
}

template<class Item>
void Queue2<Item>::print()
{
	input.print();
}

#endif
```

Explanation:

To simulate a queue using 2 stacks, I made a new class called Queue2. Its privates are 2 stacks, from the stack class I made, used and capacity. The size and empty function uses the same algorithms as our stack and queue. Size is return used, and empty is check if used it 0. I have stack 1 as our input holder stack, and stack 2 as our reverse stack to mimic a queue. We push things into our Queue2, the same way we input into our stacks so the code is just to input into s1. For our pop functions, we need to replicate a Queue which does FIFO, while using 2 stacks. Since 1 stack holds all user inputs at the moment, we pop and push everything into the other stack. So if user inputted 1,2,3, s1 hold 1,2,3 but a Queue would in reality pop the front which is 1, but a stack can only pop the back which is 3, so we reverse it by popping and pushing into our s2, so now s2 holds 3,2,1 and now we have access to the 1 which was the users first input, so we pop it, do used-- and then put it back into s1, so s2 is 1,2. Front works the same way, where we can't access the 1 that the user first inputted so we have to reverse it and flip it to s2, so s2 is 3,2,1, store top of s2 into a temp variable, put s2 back into s1, and return the temp variable. Our

print function which technically shouldn't exist was just using my stack print function which is just coutting the for loop of our dynamic array.

For question 5, here is my header file:

```
#ifndef __ZAMAN_FARHAN_HW4_Q5_H__
#define __ZAMAN_FARHAN_HW4_Q5_H__

#include "Zaman_Farhan_HW4_Q2.h"

template<class Item>
class Stack2
{
	private:
		Queue<Item> input;
		Queue<Item> temp;
		size_t count;
		size_t cap;
	public:
		Stack2(size_t initSize=30);
		~Stack2();
		void push(const Item& entry);
		void pop();
		bool empty() const {return(count==0);}
		size_t size() const {return count;}
		Item top();
		void print();
};

#include "Zaman_Farhan_HW4_Q5.cpp"
#endif
```

For question 5, here is my cpp file:

```
#ifndef __ZAMAN_FARHAN_HW4_Q5_CPP__
#define __ZAMAN_FARHAN_HW4_Q5_CPP__

#include "Zaman_Farhan_HW4_Q5.h"

template<class Item>
Stack2<Item>::Stack2(size_t initSize)
{
	count=0;
	cap=initSize;
}
```

```
template<class Item>
Stack2<Item>::~Stack2()
{
	for(int i=0; i<count; i++)
	{
		input.pop();
	}
}

template<class Item>
void Stack2<Item>::push(const Item& entry)
{
	input.push(entry);
	count++;
}

template<class Item>
void Stack2<Item>::pop()
{
	for(int i=0; i<count-1; i++)
	{
		temp.push(input.front());
		input.pop();
	}
	input.pop();
	count--;
	for(int i=0; i<count; i++)
	{
		input.push(temp.front());
		temp.pop();
	}
}

template<class Item>
Item Stack2<Item>::top()
{
	for(int i=0; i<count-1; i++)
	{
		temp.push(input.front());
		input.pop();
	}
	Item store=input.front();
	input.pop();
	temp.push(store);
	for(int i=0; i<count; i++)
	{
		input.push(temp.front());
```

```
            temp.pop();
      }
      return store;
}

template<class Item>
void Stack2<Item>::print()
{
      input.print();
}

#endif
```

Explanation:

To mimic a stack using 2 queues, I made a class called Stack2. Similarly to the previous one, I have privates of 2 queues, used and capacity. The size function is just return used, and the empty function is just check if used is 0 and return true or false. My push function is the same thing as a push in queue, so that was just push into q1 and used++. For pop we have to mimic a stack which would pop the back element, so in our q1, we intake user inputs, let's say it is 1,2,3. To mimic a Stack, we need it to delete 3, but our Queue, only has access to 1, so since we can't reverse it and do the same thing as before, we instead put 1,2 into our q2, so q1 now has 3, and q2 has 1,2. We pop q1 so q1 is empty, and then push q2 into q2 so now q1 has 1,2 and q2 has nothing. Similarly for our top function, to mimic the top of a stack which would be 3 for this case, we have to push up to used-1 into q2, so only the last element is left in q1, so q2 has 1,2 and q1 has 3. We assign into a temp the value of q1, and then clear q1, so its empty, push the variable into q2, so its in order and q2 is 1,2,3, and then push q2 into q1, and clear q2, and return the temp. My print function which technically doesn't exist just calls the print function of queue, which is print out in order of the array from first to last.

Improvements:

I think the only improvements to make are using the resize function to properly allocate memory, making copy constructor, destructor, assignment operator. There should be a better way to print the whole stack or queues, so stack every chance can be seen to help user or anyone else visualize it. Also a queue is circular array so there are cases of first>=last and last being negative that need to be dealt with.