Homework 5

Mengxiang Jiang
CSEN 5303 Foundations of Computer Science

September 19, 2022

Problem 1 (Stacks). Let S1 and S2 be two stacks.
1. Is it possible to keep two stacks in a single array, if one grows from position 1 of the array, and the other grows from the last position?
2. Write a procedure Push(x, S) that pushes element x onto stack S, where S is one or the other of these two stacks. Include all necessary error checks in your procedure.

1. Yes, as long as the two stacks don't grow to occupy a bigger size than the array when added together, then there shouldn't be a problem.
2.

```
procedure Push(x: item, S: Stack);
    var
        head_idx, tail_idx: integer;
        head_type: boolean;
    begin
        head_idx := S.head_idx;
        tail_idx := S.tail_idx;
        if head_idx + 1 >= tail_idx then
            throw StackFullError;
        else
            begin
                if head_type = true then
                    begin
                        S.head_idx := head_idx + 1;
                        S.array[S.head_idx] := x;
                    end;
                else
                    begin
                        S.tail_idx := tail_idx - 1;
                        S.array[S.tail_idx] := x;
                    end;
            end;
    end;
```

Problem 2 (Stacks). Consider the fundamental theorem of arithmetic, which is stated as follows: Every positive integer greater than 1 can be written uniquely as a prime or as the product of two or more primes, where the prime factors are written in order of nondecreasing size. We want to use a stack to read a number and print all of its prime divisors in descending order. For example, with the integer 2100, the output should be:

7 5 5 3 2 2

1. Write an algorithm, called *Prime_Factorization*, which accepts a positive integer greater than 1, and generates its prime factorization according to the above-mentioned theorem. [*Hint:* The smallest divisor greater than 1 of any integer is guaranteed to be a prime.]

```
function Prime_Factorization(n: integer) : StackInt;
    var
        factors : StackInt;
        p : integer;
    begin
        p := 2;
        while n > 1 do
            begin
                if n mod p = 0 then
                    begin
                        factors.push(p);
                        n := n / p;
                    end;
                else
                    p := p + 1;
            end;
        return factors;
    end;
```

2. Propose a stack class to accommodate this prime decomposition. It should have at least two member functions: one to compute the prime factorization of an integer, and one to print all corresponding prime divisors in descending order.

```
class PrimeFactorizationStack
    private factors_stack
    private n

    public procedure new(n)
    // initialize variables of PrimeFactorizationStack

    private procedure prime_factorization()
    // fill the factors_stack with the prime factors

    public procedure print_prime_factors()
    // call prime_decomposition then use the stack to print the factors
```

3. Give an implementation of all member functions defined in the above stack class.

```
procedure prime_factorization();
    var
        n, p : integer;
    begin
        if self.factors_stack.is_empty() = true then
            begin
                self.factors_stack = new StackInt;
                n := self.n;
                p := 2;
                while n > 1 do
                    begin
                        if n mod p = 0 then
                            begin
                                factors.push(p);
                                n := n / p;
                            end;
                        else
                            p := p + 1;
                    end;
            end;
    end;

procedure print_prime_factors();
    var
        p : integer;
    begin
        if self.factors_stack.is_empty() = true then
            self.prime_factorization();
        while (self.factors_stack.is_empty() = false) do
            begin
                p := self.factors_stack.top();
                print(p.to_str() + " ");
                self.factors_stack.pop();
            end
    end
```

Problem 3 (Stacks). Write a segment of code to perform each of the following operations. You may call any of the member functions of Stack Type. The details of the stack type are encapsulated; you may use only the stack operations in the specification to perform the operations. (You may declare additional stack objects).

a. Set *secondElement* to the second element in the stack, leaving the stack without its original top two elements.

```
//assuming we are given s1 as the stack
s1.pop();
ItemType secondElement = s1.top();
s1.pop();
```

b. Set *bottom* equal to the bottom element in the stack, leaving the stack empty.

```
//assuming we are given s1 as the stack
ItemType bottom;
while (s1.isEmpty() == false) {
    bottom = s1.top();
    s1.pop();
}
```

c. Set *bottom* equal to the bottom element in the stack, leaving the stack unchanged.

```
//assuming we are given s1 as the stack
ItemType bottom;
StackType s2 = new StackType();
while (s1.isEmpty() == false) {
    bottom = s1.top();
    s1.pop();
    s2.push(bottom);
}
while (s2.isEmpty() == false) {
    s1.push(s2.top());
    s2.pop();
}
```

d. Make a copy of the stack, leaving the stack unchanged.

```
//assuming we are given s1 as the stack
ItemType temp;
//s3 is the copy, s2 is used to help
StackType s2 = new StackType();
StackType s3 = new StackType();
while (s1.isEmpty() == false) {
    s2.push(s1.top());
    s1.pop();
}
while (s2.isEmpty() == false) {
    temp = s2.top()
    s1.push(temp);
    s3.push(temp);
    s2.pop();
}
```