

Midterm

Note

This is a closed-book exam. Each problem accounts for 10 points, unless otherwise marked.

Problems

1. Explain the following terms:
 - (a) abstract data type (ADT)
 - (b) precondition and postcondition (of a function/method)
2. Consider the ADT polynomial for integer polynomials over a single variable x , which includes the following operations:
 - `+degree(): int`
Returns the degree of a polynomial
 - `+coefficient(n: int): int`
Returns the coefficient of x^n .
 - `+changeCoefficient(newCoeff: int, n: int): bool`
Replaces the coefficient of x^n with `newCoeff`, and returns *true* if the operation is carried out successfully and *false* otherwise.
 - (a) Using these operations, write statements (in pseudocode) to compute the sum of two polynomials.
 - (b) Can one find out the highest possible degree of a polynomial supported by a particular implementation of the ADT? How?
3. The following is a recursive, but simplified, variant of Euclid's algorithm for computing the greatest common divisor of two positive integers.

```
int gcd(int a, int b)
{
    if (a == 0)
        return b;
    if (b == 0)
        return a;
    if (a > b)
        return gcd(a - b, b);
    else
        return gcd(a, b - a);
}
```

- (a) What are the base cases? Why are they appropriate?
 - (b) Will the base cases always be reached? Why?
4. Consider the array-based implementation of the ADT bag, the C++ class `ArrayBag`. We have implemented the function `remove()` with a static array as follows:

```
template<typename ItemType>
bool ArrayBag::remove(const typename& anEntry)
{
    int locatedIndex = getIndexOf(anEntry);
    bool canRemoveItem = (locatedIndex > -1);
    if(canRemoveItem)
    {
        itemCount--;
        items[locatedIndex] = items[itemCount];
    }
    return canRemoveItem;
}
```

Suppose that we now want to implement `ArrayBag` with dynamic memory allocation. The class definition has been changed to

```
template<typename ItemType>
class ArrayBag: public BagInterface
{
private:
    static const int DEFAULT_CAPACITY = 6;
    ItemType* items;
    int itemCount;
    int maxItems;
    int getIndexOf(const ItemType& target) const;
public:
    // public member functions are omitted
};
```

where `items` is a pointer pointing to the dynamic array. We now want to modify `remove()` and add the following feature: *When the number of items is no greater than half of the current bag capacity, cut down the bag capacity to its half.* Rewrite `remove()` to add this feature.

Note. You may (1) assume that the bag capacity will always be an odd number, and (2) use the function `getIndexOf()` to obtain the array index of a given item if it exists or `-1` otherwise.

5. Consider the link-based implementation of the ADT bag, the C++ class `LinkedBag`. We have implemented the function `add()`, which adds an item at the beginning position of our bag, as follows:

```

template<class ItemType>
bool LinkedBag<ItemType>::add(const ItemType& newEntry)
{
    Node<ItemType>* newNodePtr = new Node<ItemType>();
    newNodePtr->setItem(newEntry);
    newNodePtr->setNext(headPtr);
    headPtr = newNodePtr;
    itemCount++;
    return true;
}

```

Rewrite this function to add an item at the *ending* position of our bag.

6. When we implemented the classes `LinkedBag`, we used C++ templates to make our classes capable of storing all types of items. With this in mind, please answer the following questions.

- (a) What will be the output of the following program?

```

#include <iostream>
using namespace std;

template<typename A, typename B>
void f(A a, B b)
{
    cout << a + b << endl;
}

int main()
{
    f<double, double>(1.23, 10.3);
    f<double, int>(1.23, 10.3);
    f<int, double>(1.23, 10.3);
    f<char, char>(1.23, 10.3);
    return 0;
}

```

- (b) Suppose that we have two classes A and B, and want to use one bag to store items of both classes. Propose a way to modify `LinkedBag` or A and B to complete the task.
7. Convert the infix expression $(a / (b / c)) + d - e * f$ to the postfix form. Please follow the conversion algorithm discussed in class (see the appendix), and show the status of the stack and the current postfix expression after each character of the infix expression is processed.
8. Consider again the infix to postfix conversion algorithm. Suppose we want to allow the exponent operator $^$, which has higher precedence than $*$ and $/$ and is associated

to the right, i.e., $a \wedge b \wedge c$ equals $a \wedge (b \wedge c)$. Please modify the conversion algorithm to allow the additional \wedge .

9. How are stacks and recursion related? Please explain the relationship by considering solutions to the task of determining the existence of a path on a directed graph.
10. Design an algorithm (in pseudocode) to sort a stack of integers in increasing order with the smallest element on the top. You may use additional stacks and integer variables, but no other data structures (including arrays). You should try to use as few additional stacks as possible.

Appendix

- Below is the algorithm discussed in class for converting an infix expression to a postfix expression.

```

for (each character ch in the infix expression) {
    switch (ch) {
        case operand: // Append operand to the end of postfixExp
            postfixExp = postfixExp + ch
            break
        case '(':      // Save '(' on the stack aStack
            aStack.push(ch)
            break
        case operator: // Process operators of higher precedence
            while (!aStack.isEmpty() and
                aStack.peek() is not a '(' and
                precedence(ch) <= precedence(aStack.peek())) {
                postfixExp = postfixExp + aStack.peek()
                aStack.pop()
            }
            aStack.push(ch) // Save the operator
            break
        case ')':      // Pop the stack until matching '('
            while (aStack.peek() is not a '(') {
                postfixExp = postfixExp + aStack.peek()
                aStack.pop()
            }
            aStack.pop() // Remove the matching '('
            break
    }
}
// Append to postfixExp the operators remaining in the stack
while (!aStack.isEmpty()) {
    postfixExp = postfixExp + aStack.peek()
    aStack.pop()
}

```