BÁCH KHOA E-LEARNING

Started on	Sunday, 3 October 2021, 12:45 PM
State	Finished
Completed on	Sunday, 3 October 2021, 2:45 PM
Time taken	1 hour 59 mins
Marks	3.20/7.00
Grade	**4.57** out of 10.00 (**46**%)

Question **1**

Partially correct

Mark 0.20 out of 1.00

Polynomials is an important application of arrays and linked lists. A polynomial is composed of different terms where each of them holds a coefficient and an exponent. A polynomial p(x) is the expression in variable x which is in the form ($a_nx^n + a_{n-1}x^{n-1} + ... + a_1x + a_0$), where a_n, a_{n-1}, ..., a_0 fall in the category of real numbers and 'n' is the non-negative integer, which is called the degree of polynomial.

Example: $10x^2 + 26x$, here 10 and 26 are coefficients and 2, 1 is its exponential value.

Points to keep in Mind while working with Polynomials:
- The sign of each coefficient and exponent is stored within the coefficient and the exponent itself.
- The storage allocation for each term in the polynomial must be done in descending order of their exponent.

In this question, complete SLinkedList class is included. You should use this class to complete your Polynomial class with initialized frame as following. This task is implement insertTerm to insert a term into a polynomial.

```
class Polynomial;
class Term {
private:
    double coeff;
    int exp;
    friend class Polynomial;
public:
    Term(double coeff = 0.0, int exp = 0) {
        this->coeff = coeff;
        this->exp = exp;
    }
    bool operator==(const Term& rhs) const {
        return this->coeff == rhs.coeff && this->exp == rhs.exp;
    }
    friend ostream & operator<<(ostream &os, const Term& term) {
        cout << endl;
        cout << "Term: " << "(" << term.coeff << " " << term.exp << ")";
        return os;
    }
};
class Polynomial {
private:
    SLinkedList<Term>* terms;
public:
    Polynomial() {
        this->terms = new SLinkedList<Term>();
    }
    ~Polynomial() {
        this->terms->clear();
    }
    void insertTerm(const Term& term);
    void insertTerm(double coeff, int exp);
    void print() {
        SLinkedList<Term>::Iterator it;
        cout << "[";
        for (it = this->terms->begin(); it != this->terms->end(); it++) {
            cout << (*it);
        }
        cout << endl << "]";
    }
};
```

For example:

Test	Result

`Polynomial *poly = new Polynomial();`
`poly->insertTerm(6.0, 2);`
`poly->insertTerm(4.0, 5);`
`poly->insertTerm(4.0, 3);`
`poly->insertTerm(6.0, 5);`
`poly->insertTerm(-1.0, 0);`
`poly->insertTerm(-6.0, 6);`
`poly->insertTerm(6.0, 6);`
`poly->print();`
	`[`
	`Term: (10 5)`
	`Term: (4 3)`
	`Term: (6 2)`
	`Term: (-1 0)`
	`]`

Answer: (penalty regime: 0 %)

Reset answer

```
void Polynomial::insertTerm(const Term& term) {
    // STUDENT ANSWER
}

void Polynomial::insertTerm(double coeff, int exp) {
    // STUDENT ANSWER
}
```

	Test	Expected	Got	

✗	`Polynomial *poly = new Polynomial();`
	`poly->insertTerm(6.0, 2);`
	`poly->insertTerm(4.0, 5);`
	`poly->insertTerm(4.0, 3);`
	`poly->insertTerm(6.0, 5);`
	`poly->insertTerm(-1.0, 0);`
	`poly->insertTerm(-6.0, 6);`
	`poly->insertTerm(6.0, 6);`
	`poly->print();`
		[
		Term: (10 5)
		Term: (4 3)
		Term: (6 2)
		Term: (-1 0)
		]
			[
			]
				✗

✗	`int n = 5;`
	`int coeff[] = { 1, 2, 3, 4, 5 };`
	`int exp[] = { 1, 2, 3, 4, 5 };`
	
	`Polynomial p1;`
	`for (int i = 0; i < n; ++i)`
	`  p1.insertTerm(coeff[i], exp[i]);`
	
	`p1.print();`
		[
		Term: (5 5)
		Term: (4 4)
		Term: (3 3)
		Term: (2 2)
		Term: (1 1)
		]
			[
			]
				✗

✗	`int n = 5;`
	`int coeff[] = { 1, -1, 3, 4, 5 };`
	`int exp[] = { 1, 1, 3, 4, 5 };`
	
	`Polynomial p1;`
	`for (int i = 0; i < n; ++i)`
	`  p1.insertTerm(coeff[i], exp[i]);`
	
	`p1.print();`
		[
		Term: (5 5)
		Term: (4 4)
		Term: (3 3)
		]
			[
			]
				✗

✓	`int n = 5;`
	`int coeff[] = { 0, 0, 0, 0, 0 };`
	`int exp[] = { 1, 1, 3, 4, 5 };`
	
	`Polynomial p1;`
	`for (int i = 0; i < n; ++i)`
	`  p1.insertTerm(coeff[i], exp[i]);`
	
	`p1.print();`
		[
		]
			[
			]
				✓

	Test	Expected	Got	

✖	`int n = 5;`
	`int coeff[] = { 1, 0, 0, 0, 0 };`
	`int exp[] = { 0, 1, 3, 4, 5 };`
	
	`Polynomial p1;`
	`for (int i = 0; i < n; ++i)`
	`  p1.insertTerm(coeff[i], exp[i]);`
	
	`p1.print();`
		[
		Term: (1 0)
		]
			[
			]
				✖

Show differences

Partially correct

Marks for this submission: 0.20/1.00.

Question **2**

Incorrect

Mark 0.00 out of 1.00

Given the head of a doubly linked list, two positive integer a and b where a <= b. Reverse the nodes of the list from position a to position b and return the reversed list

Note: the position of the first node is 1. It is guaranteed that a and b are valid positions. You MUST NOT change the val attribute in each node.

```
struct ListNode {
    int val;
    ListNode *left;
    ListNode *right;
    ListNode(int x = 0, ListNode *l = nullptr, ListNode* r = nullptr) : val(x), left(l), right(r) {}
};
```

Constraint:

1 <= list.length <= 10^5

0 <= node.val <= 5000

1 <= left <= right <= list.length

Example 1:

Input: list = {3, 4, 5, 6, 7} , a = 2, b = 4

Output: 3 6 5 4 7

Example 2:

Input: list = {8, 9, 10}, a = 1, b = 3

Output: 10 9 8

For example:

Test	Input	Result

int size;
cin >> size;
int* list = new int[size];
for(int i = 0; i < size; i++) {
cin >> list[i];
}
int a, b;
cin >> a >> b;
unordered_map<ListNode*, int> nodeValue;
ListNode* head = init(list, size, nodeValue);
ListNode* reversed = reverse(head, a, b);
try {
printList(reversed, nodeValue);
}
catch(char const* err) {
cout << err << '\n';
}
freeMem(head);
delete[] list;
	5
	3 4 5 6 7
	2 4
		3 6 5 4 7

Test	Input	Result

int size;
cin >> size;
int* list = new int[size];
for(int i = 0; i < size; i++) {
cin >> list[i];
}
int a, b;
cin >> a >> b;
unordered_map<ListNode*, int> nodeValue;
ListNode* head = init(list, size, nodeValue);
ListNode* reversed = reverse(head, a, b);
try {
printList(reversed, nodeValue);
}
catch(char const* err) {
cout << err << '\n';
}
freeMem(head);
delete[] list;
	3
	8 9 10
	1 3
		10 9 8

Answer: (penalty regime: 0 %)

Reset answer

```
/*
struct ListNode {
    int val;
    ListNode *left;
    ListNode *right;
    ListNode(int x = 0, ListNode *l = nullptr, ListNode* r = nullptr) : val
};
*/

  typedef ListNode* ptr;
ListNode* reverse(ListNode* head, int a, int b) {
    int idx = 0;
    ptr b_ptr = head;
    while (b_ptr != NULL)
    {
        if (idx == b) break;
        b_ptr = b_ptr->right;
        ++idx;
    }
    ptr a_ptr = head;
    idx = 0;
     while (a_ptr != NULL)
    {
        if (idx == a) break;
        a_ptr = a_ptr->right;
        ++idx;
    }
     a_ptr->left->right = b_ptr;
    for (int i = b; i >= a; --i)
    {
        b_ptr->right = b_ptr->left;
        b_ptr = b_ptr->left;
}
    return head;
}
```

	Test	Input	Expected	Got	
✖	`int size;` `    cin >> size;` `    int* list = new int[size];` `    for(int i = 0; i < size; i++) {` `        cin >> list[i];` `    }` `    int a, b;` `    cin >> a >> b;` `unordered_map<ListNode*, int> nodeValue;` `    ListNode* head = init(list, size, nodeValue);` `    ListNode* reversed = reverse(head, a, b);` `    try {` `printList(reversed, nodeValue);` `    }` `    catch(char const* err) {` `        cout << err << '\n';` `    }` `    freeMem(head);` `    delete[] list;`	5 3 4 5 6 7 2 4	3 6 5 4 7	3 4 7 6 5 ...snip... 5 4 7 6 5 4 7 6 5 4 7 6 5	✖

	Test	Input	Expected	Got	
				4 7 6 5 4 7 6 ***Time limit exceeded***	

Testing was aborted due to error.

Show differences

Incorrect

Marks for this submission: 0.00/1.00.

Question **3**

Correct

Mark 1.00 out of 1.00

Given a string **S** of characters, a *duplicate removal* consists of choosing two adjacent and equal letters, and removing them.

We repeatedly make duplicate removals on **S** until we no longer can.

Return the final string after all such duplicate removals have been made.

For example:

Test	Result
`cout << removeDuplicates("abbaca");`	`ca`
`cout << removeDuplicates("aab");`	`b`

Answer: (penalty regime: 0, 0, 5, 10 %)

Reset answer

```
string removeDuplicates (string S)
{
    int size = S.size();
    stack<char> stk;
    for (int i = 0; i < size ;++i)
    {
        if (stk.empty() || S[i] != stk.top()) stk.push(S[i]);
        else stk.pop();
    }
    string res = "";
    while (!stk.empty())
    {
        res = stk.top() + res;
        stk.pop();
    }
    return res;
}
```

	Test	Expected	Got	
✔	`cout << removeDuplicates("abbaca");`	`ca`	`ca`	✔
✔	`cout << removeDuplicates("aab");`	`b`	`b`	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **4**

Correct

Mark 1.00 out of 1.00

Given a string **s** containing just the characters **'(', ')', '[', ']', '{', and '}'**. Check if the input string is valid based on following rules:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

For example:

- String **"[]()"** is a valid string, also **"[()]".**
- String "**[])**" is **not** a valid string.

Your task is to implement the function

```
bool isValidParentheses (string s){
    /*TODO*/
}
```

For example:

Test	Result
`cout << isValidParentheses("[]");`	1
`cout << isValidParentheses("[]()");`	1
`cout << isValidParentheses("[)");`	0

Answer: (penalty regime: 0, 0, 5, 10 %)

Reset answer

```
bool isValidParentheses(string s)
{
    stack<char> stk;
    char x;

    // Traversing the Expression
    for (int i = 0; i < (int)s.length(); i++)
    {
        if (s[i] == '(' || s[i] == '['
            || s[i] == '{')
        {
            // Push the element in the stack
            stk.push(s[i]);
            continue;
        }

        // IF current current character is not opening
        // bracket, then it must be closing. So stack
        // cannot be empty at this point.
        if (stk.empty())
            return false;

        switch (s[i]) {
        case ')':

            // Store the top element in a
            x = stk.top();
            stk.pop();
            if (x == '{' || x == '[')
                return false;
            break;

        case '}':

```

```
            // Store the top element in b
            x = stk.top();
            stk.pop();
            if (x == '(' || x == '[')
                return false;
            break;

        case ']':

            // Store the top element in c
            x = stk.top();
            stk.pop();
            if (x == '(' || x == '{')
                return false;
            break;
        }
    }

    // Check Empty Stack
    return (stk.empty());
}
```

	Test	Expected	Got	
✔	`cout << isValidParentheses("[]()");`	1	1	✔
✔	`cout << isValidParentheses("[)");`	0	0	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **5**

Correct

Mark 1.00 out of 1.00

Given an array nums[] of size N, the task is to find the next greater element for each element of the array
Next greater element of an element in the array is the nearest element on the right which is greater than the current element.
If there does not exist a next greater of a element, the next greater element for it is -1

Note: iostream, stack and vector are already included

Constraints:
1 <= nums.length <= 10^5
0 <= nums[i] <= 10^9

Example 1:
Input:
nums = {15, 2, 4, 10}
Output:
{-1, 4, 10, -1}

Example 2:
Input:
nums = {1, 4, 6, 9, 6}
Output:
{4, 6, 9, -1, -1}

For example:

Test	Input	Result

`int N;`
`cin >> N;`
`vector<int> nums(N);`
`for(int i = 0; i < N; i++) cin >> nums[i];`
`vector<int> greaterNums = nextGreater(nums);`
`for(int i : greaterNums)`
`  cout << i << ' ';`
`cout << '\n';`
	4
	15 2 4 10
		-1 4 10 -1

`int N;`
`cin >> N;`
`vector<int> nums(N);`
`for(int i = 0; i < N; i++) cin >> nums[i];`
`vector<int> greaterNums = nextGreater(nums);`
`for(int i : greaterNums)`
`  cout << i << ' ';`
`cout << '\n';`
	5
	1 4 6 9 6
		4 6 9 -1 -1

Answer: (penalty regime: 0 %)

Reset answer

```
vector<int> nextGreater(vector<int>& arr){
    stack<int> stk;
    int size = arr.size();
    for (int i  = size -1 ; i >= 0;  --i) stk.push(arr[i]);
    for (int i = 0; i < size- 1; ++i)
    {
```

```
        for (int j = i+1; j < size; ++j)
        {
            if (arr[j] > stk.top() ) {arr[i] = arr[j]; break;}
            else
            {
                if (j == size -1 ) arr[i] = -1;
            }
        }
        stk.pop();
    }
    arr[arr.size()-1] = -1;
    return arr;
}
```

	Test	Input	Expected	Got	

✔	`int N;`
	`cin >> N;`
	`vector<int> nums(N);`
	`for(int i = 0; i < N; i++) cin >> nums[i];`
	`vector<int> greaterNums = nextGreater(nums);`
	`for(int i : greaterNums)`
	`  cout << i << ' ';`
	`cout << '\n';`
		4
		15 2 4 10
			-1 4 10 -1	-1 4 10 -1	✔

✔	`int N;`
	`cin >> N;`
	`vector<int> nums(N);`
	`for(int i = 0; i < N; i++) cin >> nums[i];`
	`vector<int> greaterNums = nextGreater(nums);`
	`for(int i : greaterNums)`
	`  cout << i << ' ';`
	`cout << '\n';`
		5
		1 4 6 9 6
			4 6 9 -1 -1	4 6 9 -1 -1	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **6**

Incorrect

Mark 0.00 out of 1.00

Research **queue** which is implemented in C library at: http://www.cplusplus.com/reference/queue/queue/. You can use library **queue** in c++ for this question.

Using **queue**, complete function **void bfs(vector<vector<int>> graph, int start)** to traverse all the nodes of the graph from given start node using Breadth First Search algorithm and data structure **queue**, and print the order of visited nodes.

You can use below liberaries in this question.

```
#include <iostream>
#include <vector>
#include <queue>
```

For example:

Test	Result

`int init_graph[10][10] = { {0, 1, 1, 0, 1, 0, 1, 0, 1, 0},`
`{0, 0, 1, 1, 0, 0, 0, 1, 0, 0},`
`{0, 1, 0, 0, 0, 1, 1, 0, 1, 1},`
`{1, 0, 0, 0, 0, 0, 0, 1, 0, 0},`
`{0, 1, 0, 0, 0, 0, 0, 1, 0, 0},`
`{1, 0, 1, 0, 1, 0, 0, 0, 1, 0},`
`{0, 0, 1, 1, 0, 1, 0, 0, 0, 0},`
`{1, 0, 0, 0, 0, 1, 1, 0, 1, 0},`
`{0, 0, 0, 0, 0, 1, 0, 1, 0, 1},`
`{1, 0, 1, 0, 1, 0, 0, 0, 1, 0} };`
`int n = 10;`
`vector<vector<int>> graph(n, vector<int>());`
`for (int i = 0; i < n; ++i) {`
`for (int j = 0; j < n; ++j) {`
`if (init_graph[i][j]) graph[i].push_back(j);`
`}`
`}`
 	
`bfs(graph, 0);`
	`0 1 2 4 6 8 3 7 5 9`

Answer: (penalty regime: 0 %)

Reset answer

```
1 void bfs(vector<vector<int>> graph, int start) {
2
3 }
```

	Test	Expected	

Your code failed one or more hidden tests.

Incorrect

Marks for this submission: 0.00/1.00.

Question **7**

Incorrect

Mark 0.00 out of 1.00

Research **queue** which is implemented in C library at http://www.cplusplus.com/reference/queue/queue/. You can use library **queue** in c++ for this question.

Using queue, complete function **vector<int> topologicalSorting(vector<vector<int>> graph)** to find the order topology of all nodes in the graph. In case, graph doesn't have order topology, return an empty vector**.**

You can use below liberaries in this question.

```
#include <iostream>
#include <vector>
#include <queue>
```

For example:

Test	Result

int n = 6;
int G[6][6] = { {0, 1, 0, 1, 0, 0},
{0, 0, 1, 1, 0, 0},
{0, 0, 0, 1, 1, 1},
{0, 0, 0, 0, 1, 1},
{0, 0, 0, 0, 0, 1},
{0, 0, 0, 0, 0, 0} };
vector<vector<int>> graph(n, vector<int>());
for (int i = 0; i < n; ++i) {
for (int j = 0; j < n; ++j) {
if (G[i][j] == 1) graph[i].push_back(j);
}
}
 	
vector<int> res = topologicalSorting(graph);
if (res.empty()) printf("-1");
else {
for (int e : res) printf("%d ", e);
}
	0 1 2 3 4 5

Test	Result

int n = 6;
int G[6][6] = { {0, 1, 0, 1, 0, 0},
{0, 0, 1, 1, 0, 0},
{0, 0, 0, 1, 1, 1},
{0, 0, 0, 0, 1, 1},
{0, 0, 0, 0, 0, 1},
{0, 1, 0, 0, 0, 0} };
vector<vector<int>> graph(n, vector<int>());
for (int i = 0; i < n; ++i) {
for (int j = 0; j < n; ++j) {
if (G[i][j] == 1) graph[i].push_back(j);
}
}
 	
vector<int> res = topologicalSorting(graph);
if (res.empty()) printf("-1");
else {
for (int e : res) printf("%d ", e);
}
	-1

Answer: (penalty regime: 0 %)

Reset answer

```
vector<int> topologicalSorting(vector<vector<int>> graph) {

}
```

Syntax Error(s)

```
__tester__.cpp: In function 'std::vector<int> topologicalSorting(std::vector<std::vector<int> >)':
__tester__.cpp:10:1: error: no return statement in function returning non-void [-Werror=return-type]
 }
 ^
cc1plus: all warnings being treated as errors
```

Incorrect

Marks for this submission: 0.00/1.00.

◀ Lab 2: Preparation

Jump to...

Lab 2: Sorting ▶