Jingchao Zhou
6361657578
EE599 HW3

Question 1

- Passing parameter by value
    - Pros: Arguments passed by can be anything and they are never changed by the function being called, which prevents side effects.
    - Cons: Copying structs and classes can incur a significant performance penalty, especially if the function is called many times.
    - When to use: when passing fundamental data type and enumerators and the function does not need to change the argument.
- Passing parameter using pointer
    - Pros: Passing parameter using pointer allows a function to change the value of the argument. It is fast even when used with large structs or classes. It also allows multiple values as a return from a function.
    - Cons: Because literals and expressions do not have addresses, pointer arguments must be normal variables. It needs to be declared and initialized every time. It is slower than passing parameters by value.
    - When to use: When passing a pointer and nullptr is a valid argument logically.
- Passing parameter using reference
    - Pros: References allow a function to change the value of the argument, which is sometimes useful. Since no copy of the argument is made, passing by reference is fast. It also can be used to return multiple values from a function. No worry about null values.
    - Cons: It's impossible to tell from the function call whether the argument may change, since an argument passed by value and passed by using reference looks the same.
    - When to use: when passing structs or classes. When you need the function to modify an argument.
- Passing parameter using constant reference
    - Pros: References allow a function to change the value of the argument, which is sometimes useful. Otherwise, const references can be used to guarantee the function won't change the argument.
    - Cons: The argument cannot be changed. The original data is not copied.
    - When to use: when passing structs or classes and read only.

Question 2

```
vector<int> Solution::twoSum(vector<int>& nums, int target) {
 int i,j;
 vector<int> result;
 for (i = 0;i<nums.size();i++){
     for(j = i+1; j<nums.size();j++){
         if (target == nums[i]+nums[j]){
             int myints[] = {i,j};
             result.assign(myints,myints+2);
             continue;}
         else
             continue;
     }
 }
 return result;
}//Time complexity = O(n^2)
```

Question 3

```
#include "solution.h"
#include "iostream"

SinglyLinkedList::SinglyLinkedList(){
head_ =  new ListNode(0);
}

SinglyLinkedList::SinglyLinkedList(const vector<int> &inputs, int i){
 head_ = new ListNode(0);
 ListNode * temp = head_;
 ListNode * record;
 for(int j = 0; j<inputs.size();j++){
   ListNode * t = new ListNode(inputs[j]);
   temp -> next = t;
   temp = temp -> next;
   if (j == i-1){
     record = t;
   }
 }

 if ( i > 0 && i<= inputs.size()){
   temp -> next  = record;
 }
 if (i == -1 || i > inputs.size() + 1){
   temp -> next = nullptr;
 }
}//Time Complexity O(n)

bool SinglyLinkedList::empty() {
 if (head_ -> next == NULL)
   {return true;}
 else {
   return false;
 }
 }//Time Complexity O(1)

int SinglyLinkedList::size(){
 int count = 0;
```

```
 ListNode * temp= new ListNode(0);
 temp = head_;
 while(temp -> next != NULL){
   count++;
   temp = temp -> next;
 }
return count;
}//Time Complexity O(n)

void SinglyLinkedList::push_back(int i){
 ListNode * temp= new ListNode(i);
 ListNode * curr;
 curr = head_;
   while (curr -> next != NULL){
     curr = curr -> next;
   }
   curr -> next = temp;

}//Time Complexity O(n)

void SinglyLinkedList::push_front(int i){
 ListNode *temp= new ListNode(i);
 temp -> next = head_ -> next;
 head_ -> next = temp;
}//Time Complexity O(1)

void SinglyLinkedList::insert_after(ListNode* p,int i){
 ListNode *temp= new ListNode(i);
 temp -> next = p -> next;
 p -> next = temp;
}//Time Complexity O(1)

void SinglyLinkedList::erase(ListNode* p){
ListNode * temp = head_;
 while (temp -> next != p){
   temp = temp -> next;
 }
 temp -> next = p -> next;
 p -> next = NULL;
```

```
}//Time Complexity O(n)

void SinglyLinkedList::pop_front(){
 head_ = head_ -> next;
}//Time Complexity O(1)

void SinglyLinkedList::pop_back(){
 ListNode * temp = head_;
 while(temp -> next -> next != NULL){
   temp = temp -> next;
 }
 temp -> next = NULL;
}//Time Complexity O(n)

int SinglyLinkedList::back(){
ListNode * temp = head_;
 while(temp -> next != NULL){
   temp = temp -> next;
 }
return temp -> val;
}//Time Complexity O(n)

int SinglyLinkedList::front(){
ListNode * temp = head_;
return temp -> next -> val;
}//Time Complexity O(1)

ListNode* SinglyLinkedList::GetBackPointer(){
ListNode * temp = head_;
 while(temp -> next != NULL){
   temp = temp -> next;
 }

return temp;
}//Time Complexity O(n)

ListNode* SinglyLinkedList::GetIthPointer(int i){
 int count = 0;
 ListNode * temp= new ListNode(0);
```

```
 temp = head_;
 while((temp -> next != NULL) && (count != i)){
   count++;
   temp = temp -> next;
 }
 return temp;
}//Time Complexity O(n)

void SinglyLinkedList::print(){
 ListNode *temp = NULL;
 temp = head_ -> next;
 cout << "{";
 while(temp != NULL){
   cout << temp -> val << " ";
   temp = temp -> next;
 }
 cout << "}";
}//Time Complexity O(n)



SinglyLinkedList::~SinglyLinkedList(){
   ListNode *temp = head_;
   while(temp -> next != NULL){
     temp = temp -> next;
     delete temp;
   }
   delete head_;
}//Time Complexity O(n)
```

Question 4

```
bool Solution::checkValidity(string input) {
 stack<char> s;
 char x;
 for (int i = 0; i<input.size();i++){
   if ((input[i] == '(' )|| (input[i] == '[' )|| (input[i] == '{' )){
     s.push(input[i]);
   }
   if (s.empty()){
     return false;
   }
   switch (input[i])
   {
     case ')':
     x = s.top();
     s.pop();
     if (x=='{' || x=='[')
     return false;
     break;

     case '}':
     x = s.top();
     s.pop();
     if (x=='(' || x=='[')
         return false;
     break;

     case ']':
     x = s.top();
     s.pop();
     if (x =='(' || x == '{')
         return false;
     break;
   }
 }
  return s.empty();
}//time complexity O(n)
```

Question 5

```
#include "solution.h"

AcadamicRecord::AcadamicRecord(){
maths = 0;
computers = 0;
physics = 0;
}//Time Complexity O(1)

AcadamicRecord::AcadamicRecord(int m, int c, int p){
maths = m;
computers = c;
physics = p;
}//Time Complexity O(1)

AcadamicRecord::AcadamicRecord(const AcadamicRecord &old){
maths = old.maths;
computers = old.computers;
physics = old.physics;
}//Time Complexity O(1)

AcadamicRecord AcadamicRecord::operator++(){
maths = maths + 10;
computers = computers + 10;
physics = physics + 10;

if (maths > 100){maths = 100;}
if (computers > 100){computers = 100;}
if (physics > 100){physics = 100;}

return *this;
}//Time Complexity O(1)

AcadamicRecord AcadamicRecord::operator++(int){
AcadamicRecord temp = *this;

maths = maths + 10;
computers = computers + 10;
physics = physics + 10;
```

```
if (maths > 100){maths = 100;}
if (computers > 100){computers = 100;}
if (physics > 100){physics = 100;}

return temp;
}//Time Complexity O(1)

AcadamicRecord AcadamicRecord::operator--(){
maths = maths - 20;
computers = computers - 20;
physics = physics - 20;

if (maths < 0){maths = 0;}
if (computers < 0){computers = 0;}
if (physics < 0){physics = 0;}

return *this;
}//Time Complexity O(1)

AcadamicRecord AcadamicRecord::operator--(int){
AcadamicRecord temp = *this;

maths = maths - 20;
computers = computers - 20;
physics = physics - 20;

if (maths < 0){maths = 0;}
if (computers < 0){computers = 0;}
if (physics < 0){physics = 0;}

return temp;
}//Time Complexity O(1)

AcadamicRecord AcadamicRecord::operator+=(int a){
maths = maths + a;
computers = computers + a;
physics = physics + a;
```

```
if (maths > 100){maths = 100;}
if (computers > 100){computers = 100;}
if (physics > 100){physics = 100;}

return *this;
}//Time Complexity O(1)

AcadamicRecord AcadamicRecord::operator-=(int b){
maths = maths - b;
computers = computers - b;
physics = physics - b;

if (maths < 0){maths = 0;}
if (computers < 0){computers = 0;}
if (physics < 0){physics = 0;}

return *this;
}//Time Complexity O(1)

AcadamicRecord::~AcadamicRecord(){

}//Time Complexity O(1)

string AcadamicRecord::print(){
 string m = "Maths:: " + to_string(maths);
 string c = "Computers:: " + to_string(computers);
 string p = "Physics:: " + to_string(physics);
 return m + "\n" + c + "\n" + p;
}//Time Complexity O(1)
bool AcadamicRecord::operatorEqu(AcadamicRecord &old1,AcadamicRecord &old2){
 if ((old1.maths == old2.maths) && (old1.computers == old2.computers) && (old1.physics
== old2.physics)){
   return true;
 }
 else{
   return false;
 }
}//Time Complexity O(1)
```

Question 6

```
string Solution::print(vector<int> input) {
 string a = {};
 for (int i = 0; i < input.size();i++){
   a = a + to_string(input[i]) + " ";
 }
 return a;
}//Time complexity O(n)

int Solution::first(vector<int> input){
vector<int>::iterator ptr = input.begin();
return *ptr;
}//Time complexity O(1)

int Solution::last(vector<int> input){
vector<int>::iterator ptr = input.end() - 1;
return *ptr;
}//Time complexity O(1)

int Solution::findElement(vector<int> input,int curr,int i){
 vector<int>::iterator ptr1 = find(input.begin(),input.end(),curr);
 advance(ptr1,i);
return *ptr1;
}//Time complexity O(n)
```

```
int main() {
 Solution s;
 int c = 0;
 int i = 0;
 int curr = 0;
 int index = 0;
 vector <int> input = {1,4,5,23,100,12,18,175};
 while(c != 5){
 cout << "Vector: " << s.print(input);
 cout << "Please choose any of the following options: " << endl;
 cout << "     1. What is the first element?" << endl;
 cout << "     2. What is the last element?" << endl;
 cout << "     3. What is the current element?" << endl;
 cout << "     4. What is the ith element from the current location?" << endl;
 cout << "     5. Exit." << endl;

 cin >> c;
 switch(c){
   case 1: cout << "Output: " << s.first(input) << endl;
           curr = s.first(input);
           break;
   case 2: cout << "Output: " << s.last(input) << endl;
           curr = s.last(input);
           break;
   case 3: cout << "Output: " << curr << endl;
           break;
   case 4: cout << "Enter the value of i:: "<< endl;
           cin >> i;
           while(i<0){
             cout << "i cannot be negative, please enter again" << endl;
             cin >> i;
           }
           index = distance(input.begin(),find(input.begin(),input.end(),curr));//Time
complexity O(n)
           if((index + i) > (input.size()-1)){
             cout << "Sorry! You cannot traverse " << i << " elements from your
current location" << endl;
             break;
           }
```

```
            cout << "Output: " << s.findElement(input,curr,i) << endl;
            curr = s.findElement(input,curr,i);//Time complexity O(n)
            index = 0;
            break;
    case 5: cout << "Exit!" << endl;
            break;
    }
 }
 return EXIT_SUCCESS;
}//Time complexity O(n)
```