Data Structures 2012 Fall

Practice Session 7

Contents

- STL list example
- STL stack example (Carpark example)
- Stack implementation using STL list data structure

1-STL List Example

- Creation of a list of integers:

```
int main() {
    const int SIZE = 4;
    int array[ SIZE ] = { 2, 6, 4, 8 };
    list <int> list1,list2; // create two integer lists
```

Inserting elements into list1

```
// insert items in list1
list1.push_front(1);  // into the beginning of list
list1.push_front(2);
list1.push_back(4);   // into the end of list
list1.push_back(3);

cout << "list1 contains: ";
printList(list1);
```

list1:

1			
2	1		
2	1	4	
2	1	4	3

```
// printList function to print elements in an integer list
// by using iterator to move on the list
void printList(list<int> &listRef) {
    if(listRef.empty()) // list is empty
        cout << "List is empty";
    else{
        list <int>::iterator myit;
        for(myit=listRef.begin(); myit!=listRef.end(); myit++)
            cout << *myit << " ";
    }
}
```

Sorting elements in list1

```
list1.sort(); // sort values
cout << "\nAfter sort, list1 contains: ";
printList(list1);
```

list1:

2	1	4	3	→	1	2	3	4

Inserting array elements into list2

```
// insert elements of array into list2
list2.insert(list2.begin(), array, array+SIZE);
cout << "\nAfter insert, list2 contains: ";
printList(list2);
```

list2:

2	6	4	8

Adding list2 elements at the end of list1

```
// remove elements from list2 and insert at end of list1
list1.splice(list1.end(), list2);
cout << "\nAfter splice, list1 contains: ";
printList(list1);
```

list1:

1	2	3	4

list2:

2	6	4	8

list1:

1	2	3	4	2	6	4	8

list2:

empty

Sorting elements in list1

```
list1.sort(); // sort list1
cout << "\nAfter sort, list1 contains: ";
printList(list1);
```

list1:

1	2	3	4	2	6	4	8

1	2	2	3	4	4	6	8

Inserting array elements into list2 and sorting

```
// insert elements of array into list2
list2.insert(list2.begin(), array, array+SIZE);
list2.sort(); // sort list2
cout << "\nAfter insert and sort, list2 contains: ";
printList(list2);
```

list2:

2	4	6	8

Merging list1 and list2

```
// remove elements from list2 and insert into list1 in sorted order
list1.merge(list2);
cout << "\nAfter merge:\n   list1 contains: ";
printList(list1);
cout << "\n   list2 contains: ";
printList(list2);
```

list1:

1	2	2	3	4	4	6	8

list2:

2	4	6	8

list1:

1	2	2	2	3	4	4	6	8	8

list2:

empty

Removing elements from list1

```
list1.pop_front(); // remove element from front
list1.pop_back(); // remove element from back
cout << "\nAfter pop_front and pop_back:\n   list1 contains: ";
printList(list1);
```

list1:

1	2	2	2	3	4	4	6	6	8	8

2	2	2	3	4	4	4	6	6	8

Removing duplicates

```
list1.unique(); // remove duplicate elements
cout << "\nAfter unique, list1 contains: ";
printList(list1);
```

list1:

2	2	2	3	4	4	4	6	8

2	3	4	6	8

Swapping contents of list1 and list2

```
// swap elements of list1 and list2
list1.swap(list2);
cout << "\nAfter swap:\n   list1 contains: ";
printList(list1);
cout << "\n   list2 contains: ";
printList(list2);
```

list1: 2 3 4 6 8

list2: boş

list1: empty

list2: 2 3 4 6 8

Assigning list2 elements to list1

```
// replace contents of list1 with elements of list2
list1.assign(list2.begin(), list2.end());
cout << "\nAfter assign, list1 contains: ";
printList(list1);
```

list1: empty

list2: 2 3 4 6 8

list1: 2 3 4 6 8

list2: 2 3 4 6 8

Merging list1 and list2

```
// remove list2 elements and insert into list1 in sorted order
list1.merge(list2);
cout << "\nAfter merge, list1 contains: ";
printList(list1);
```

list1: 2 3 4 6 8

list2: 2 3 4 6 8

list1: 2 2 3 3 4 4 6 6 8 8

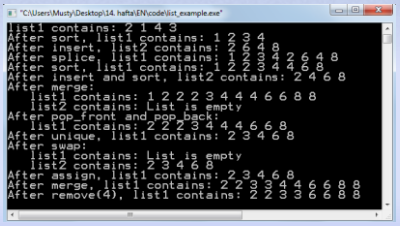
list2: empty

Removing a specified element from list

```
list1.remove(4); // remove all 4s
cout << "\nAfter remove(4), list1 contains: ";
printList(list1);
cout << endl;
```

list1: 2 2 3 3 4 4 6 6 8 8 → 2 2 3 3 6 6 8 8

Screenshot



2-Car Park Example

- There is a narrow car park where the cars can only park in a single row.
- For a car to be able to get out of the car park, all the cars behind that car have to get out first to open the way and then these cars go into the car park in the same order after the car leaves.
  - a) Write the necessary type definitions and functions to implement this car park structure (a car can be represented by using a string for its license plate)
  - b) Write a function for the nth car to leave the car park by assuming the nearest car to the exit is the 1st car.

Implementation of car park by using STL stack

- Each car is represented in the stack as a string element for the license plate.

```
// car park is implemented by using stl stack
stack <string> carpark;
// 5 cars enter the car park and contents of the car park are printed out
carpark.push("34 rst 123");
carpark.push("34 af 1234");
carpark.push("27 np 2002");
carpark.push("01 ah 1000");
carpark.push("03 bfr 7865");
cout<<"Initial status of the car park :"<<endl;
printStack(carpark);
```

Implementation of car park by using STL stack

```
// some cars leave the car park and contents of the car park are printed out
leave(3,carpark);
cout<<"Current status of the car park :"<<endl;
printStack(carpark);
leave(1,carpark);
cout<<"Current status of the car park :"<<endl;
printStack(carpark);
leave(4,carpark);
cout<<"Current status of the car park :"<<endl;
printStack(carpark);
```

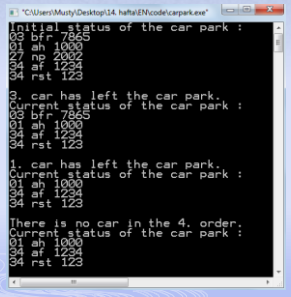
A function for the nth car to leave the car park

```
void leave(int order, stack <string> &cars){
    // a stack is used to preserve the order of the cars behind
    // after the nth car leaves
    stack <string> leftCars;
    int i;
    for(i = 0; i < order; i++){
        if(!cars.empty()){
            // the cars behind the nth car are taken into a stack
            if(i != order-1){
                leftCars.push(cars.top());
                cars.pop();
            }
            // the nth car is popped from the stack
            else{
                cars.pop();
                cout<<order<<". car has left the car park."<<endl;
            }
        }
        // stack becomes empty before reaching the nth car
        else{
            cout<<"There is no car in the "<<order<<". order."<<endl;
            i++;
            break;
        }
    }
    // the cars behind the nth car are taken back in the car park in the same order
    for(int j = 0; j < i-1; j++){
        cars.push(leftCars.top());
        leftCars.pop();
    }
}
```

Printing out stack contents on the screen

```
// a function for printing elements in a stack <string>
void printStack(stack <string> &stackRef){
    if (stackRef.empty()) // stack is empty
        cout << "Stack is empty \n";
    else{ // print stack contents
        stack <string> temp;
        // during print operation, elements in myStack are stored in another stack
        while(!stackRef.empty()){
            cout << stackRef.top() << endl;
            temp.push(stackRef.top());
            stackRef.pop();
        }
        cout << endl;
        // elements of myStack are taken back from temp to preserve contents after printing
        while(!temp.empty()){
            stackRef.push(temp.top());
            temp.pop();
        }
    }
}
```

Screenshot



3-Stack implementation using STL list data structure

- As iterators cannot be used with STL stack, a temporary stack is used to preserve contents during printing elements on the screen.
- To be able to use iterators with stack data structure, one can implement a stack by using STL list for stack elements.

Stack data structure using STL list

```
struct Stack{
    list <string> contents;
    void push(string);
    string pop();
    bool isEmpty();
    void print();
};
```

Stack functions

```
void Stack::push(string toAdd){
    contents.push_front(toAdd);  // push elements to front
}
```

```
string Stack::pop(){
    if (contents.empty()) // stack is empty
        return "\0";
    string toReturn = contents.front();
    contents.pop_front();   // pop elements from front
    return toReturn;
}
```

```
bool Stack::isEmpty(){
    return contents.empty();
}
```

Stack functions

```
void Stack::print(){
    if(contents.empty()) // stack is empty
        cout << "Stack is empty" << endl;
    else{
        // use an iterator to move on the list contents
        list <string>::iterator myit;
        for(myit=contents.begin(); myit!=contents.end(); myit++)
            cout << *myit << endl;
        cout << endl;
    }
}
```