Eli Sobylak

9-nov-15

Set 3: Exercise 3.1,3.2, 3.4, 3.5,3.6, 3.9, 3.10, 3.11, 3.22, 3.23, 3.29

3.1

```
#include <iostream>
#include <algorithm>
#include <list>
using namespace std;

int printLots(list<int> x, list<int> n) {

        list<int>::iterator k = x.begin();
        cout << "List P: " << endl;
        while(k != x.end()) {
                cout << *k << " " << endl;
                k++;
        }

        list<int>::iterator j = n.begin();
        cout << "List L: " << endl;
        while(j != n.end()) {
                cout << *j << " " << endl;
                j++;
        }

        cout << "List P on List L: " << endl;

        /*list<int>::iterator result;

        for(result = n.begin(); result != n.end(); result++) {
                if()
                cout << *result << endl;
        }
        */

        list<int>::iterator i;
        //int nPosition = distance(n.begin(), i);
        //cout << nPosition << endl;
```

```
    list<int>::iterator result;
    for(i = x.begin(); i != x.end(); i++) {
        for(result = n.begin(); result != n.end(); result++) {
            cout << *result << endl;
        }
        //result = find(n.begin(), n.end(), *i);
        //cout << *result << endl;
    }

    list<int>::iterator ptr;
    int index;

    for(index = 0, ptr = x.begin(); index < 10 && ptr != x.end(); index++, ptr++) {
        cout << *ptr << endl;
    }


    /*result = find(n.begin(), n.end(), 71;
    cout << *result << endl;
    */
    /*list<int>::iterator iter = x.begin();
    while(iter != x.end()) {
        if(*iter)
    }
    */

    return 0;
}

int main() {
    list<int> listP;
    int valueP1 = 1;
    int valueP2 = 3;
    int valueP3 = 4;
    int valueP4 = 6;
    listP.push_back (valueP1);
    listP.push_back (valueP2);
    listP.push_back (valueP3);
    listP.push_back (valueP4);


    list<int> listL;
    int valueL1 = 70;
    int valueL2 = 71;
    int valueL3 = 73;
    int valueL4 = 74;
```

```
	int valueL5 = 75;
	int valueL6 = 76;
	listL.push_back (valueL1);
	listL.push_back (valueL2);
	listL.push_back (valueL3);
	listL.push_back (valueL4);
	listL.push_back (valueL5);
	listL.push_back (valueL6);


	printLots(listP, listL);

	return 0;
}
```

3.2

```
void swap()
{
 struct node *temp=0,*nxt,*ptr;
 ptr=head;
 int count=0;
 while(ptr)
 {
   nxt=ptr->link;
   if(nxt)
 {
  if(count==0)
    head=nxt;
    count++;
   ptr->link=nxt->link;
   nxt->link=ptr;
   if(temp!=NULL)
   temp->link=nxt;
   temp=ptr;
   if(ptr->link==NULL)
   break;
   ptr=nxt->link->link;
 }
```

3.4

```
template <class InputIterator1, class InputIterator2, class OutputIterator>
  OutputIterator set_union (InputIterator1 first1, InputIterator1 last1,
                            InputIterator2 first2, InputIterator2 last2,
                    OutputIterator result)
{
 while (true)
 {
  if (first1==last1) return std::copy(first2,last2,result);
  if (first2==last2) return std::copy(first1,last1,result);

  if (*first1<*first2) { *result = *first1; ++first1; }
  else if (*first2<*first1) { *result = *first2; ++first2; }
  else { *result = *first1; ++first1; ++first2; }
  ++result;
 }
}
```

3.5

```
#include <iostream>
#include <algorithm>
#include <vector>
//#include "templateUnion.h"

using namespace std;

int main() {
    int ListA[] = {5,10,15,20,25};
    int ListB[] = {10,20,30,40,50};
    vector<int> v(10);
    vector<int>::iterator iter;

    //iter = OutputIterator
set_union<InputIterator1,InputIterator2,OutputIterator>::set_union(ListA, ListA + 5,
ListB, ListB + 5, v.begin());

    iter=std::set_difference(ListA, ListA + 5, ListB, ListB + 5, v.begin());

    v.resize(iter-v.begin());

    cout << "The union has " << (v.size()) << " elements:\n";
    for (iter=v.begin(); iter != v.end(); iter++) {
        cout << ' ' << *iter;
        cout << '\n';
    }

    return 0;
}
```

3.5

```
#include <iostream>
#include <algorithm>
#include <vector>
//#include "templateUnion.h"

using namespace std;

int main() {
        int ListA[] = {1,2,3,4,5};
        int ListB[] = {5,6,7,8,9};
        vector<int> v(10);
        vector<int>::iterator iter;

        //iter = OutputIterator
set_union<InputIterator1,InputIterator2,OutputIterator>::set_union(ListA, ListA + 5,
ListB, ListB + 5, v.begin());

        iter=std::set_union(ListA, ListA + 5, ListB, ListB + 5, v.begin());

        v.resize(iter-v.begin());

        cout << "The union has " << (v.size()) << " elements:\n";
        for (iter=v.begin(); iter != v.end(); iter++) {
                cout << ' ' << *iter;
                cout << '\n';
        }

        return 0;
}
```

3.9

Using any of these methods on a vector may invalidate a iterator looking at the vector because where I've used iterators looking at vectors I had to include a for statement that sets a beginning, end point, and step increase amounts. Normally it looks like this;

```
list<int>::iterator iter;

        for(iter = list.begin(); iter != list.end(); iter++) {

        /*code*/
}
```

3.11

```
#include <iostream>

using namespace std;

struct node {
    int info;
    struct node *next;
} *start;


class singleLink_list {
public:
    node* create_node(int);
    void search();
    void addNewValue();
    void removeValue();
    void display();
    singleLink_list() {
        start = NULL;
    }
};

int main() {
singleLink_list list;
start = NULL;

list.addNewValue();
list.addNewValue();
list.display();
list.removeValue();
list.display();
list.search();
cout << endl;
}


node *singleLink_list::create_node(int value) {
    struct node *temp, *s;
    temp = new(struct node);
    if(temp == NULL) {
```

```
                cout << "List is empty" << endl;
                return 0;
        }
        else {
                temp -> info = value;
                temp -> next = NULL;
                return temp;
        }
}

void singleLink_list::addNewValue() {
        int value;
        cout << "Enter value to be inserted" << endl;
        cin >> value;
        struct node *temp, *p;
        temp = create_node(value);
        if(start == NULL) {
                start = temp;
                start -> next = NULL;
        }
        else {
                p = start;
                start = temp;
                start -> next = p;
        }
        cout << "Element inserted at beggining" << endl;
}

void singleLink_list::display() {
        struct node *temp;
        if(start == NULL) {
                cout << "The list is empty" << endl;
                return;
        }
        temp = start;
        cout << "Elements of list are: " << endl;
        while (temp != NULL) {
                cout << temp -> info << "->";
                temp = temp -> next;
        }
        cout << "NULL" << endl;
}

void singleLink_list::search() {
        int value, pos = 0;
        bool flag = false;
        if(start == NULL) {
                cout << "List is empty" << endl;
                return;
        }
        cout << "Enter value to search for: " << endl;
        cin >> value;
        struct node *s;
        s = start;
        while(s != NULL) {
                pos++;
                if (s->info == value) {
                        flag = true;
                        cout << "Element " << value << " is found at position " << pos
<< endl;
                }
                s = s->next;
        }
        if (!flag) {
                cout << "Element " << value << " not found in the list" << endl;
        }
}

void singleLink_list::removeValue()
{
  int pos, i, counter = 0;
  if (start == NULL)
  {
    cout<<"List is empty"<<endl;
    return;
  }
  cout<<"Enter the position of value to be deleted: ";
  cin>>pos;
  struct node *s, *ptr;
  s = start;
  if (pos == 1)
  {
    start = s->next;
  }
  else
  {
    while (s != NULL)
    {
      s = s->next;
      counter++;
    }
    if (pos > 0 && pos <= counter)
    {
      s = start;
      for (i = 1;i < pos;i++)
      {
        ptr = s;
        s = s->next;
      }
      ptr->next = s->next;
    }
    else
    {
      cout<<"Position out of range"<<endl;
    }
    free(s);
    cout<<"Element Deleted"<<endl;
  }
}
```

3.22

```
#include <iostream>

using namespace std;

#define SIZE 10

      struct stack {
        stack();
        void push(char ch);
        char pop();
        int isempty();
        int peekPlace(int n);
      private:
        char stackData[SIZE];
        int topOfStack;
      };

      stack::stack()
      {
        cout << "Constructing a stack\n";
        topOfStack = 0;
      }

      void stack::push(char ch)
      {
        if(topOfStack==SIZE) {
          cout << "Stack is full\n";
          return;
        }
        stackData[topOfStack] = ch;
        topOfStack++;
      }

      char stack::pop()
      {
        if(topOfStack==0) {
          cout << "Stack is empty\n";
          return 0; // return null on empty stack
        }
        topOfStack--;
        return stackData[topOfStack];
      }

      int stack::isempty()
      {
return (topOfStack==0?1:0);
      }

      int stack::peekPlace(int n)
      {
if(!isempty())
{
  //cout << "\nElement at top is " << stackData[topOfStack-n] << endl;
  return stackData[topOfStack-n];
}
else
{
  cout << "\nStack is empty";
  return 0;
}
      }


bool IsOperand(char C)
      {
              if(C >= '0' && C <= '9') return true;
              if(C >= 'a' && C <= 'z') return true;
              if(C >= 'A' && C <= 'Z') return true;
              return false;
      }
```

```
bool IsOperator(char C)
    {
        if(C == '+' || C == '-' || C == '*' || C == '/' || C== '$')
            return true;

        return false;
    }

//int Operate(int oper,) {


//      return 0;
//}

int main() {
    stack stackObject1, stackObject2, stackObject3, stackObject4;
    int index;

    stackObject1.push('-');
    stackObject1.push('4');
    stackObject1.push('+');
    stackObject1.push('*');
    stackObject1.push('3');
    stackObject1.push('2');
    stackObject1.push('1');


    cout << "Operands? (stack 2)" << endl;
    for(index = 0; index < 7; index++) {
        int checker;
        int checked;
        checker = stackObject1.peekPlace(index+1);
        checked = IsOperand(checker);
        //cout << checker << endl;
        if(checked == 1) {
            stackObject2.push(checker);
            stackObject4.push(checker);
            //cout << "HEY" << endl;
            //cout << stackObject4.pop() << endl;
            cout << stackObject2.pop() << endl;
        }
        //cout << IsOperand(checker) << endl;
    }


    cout << "Operators? (stack 3)" << endl;
    for(index = 0; index < 7; index++) {
        int checker;
        int checked;
        //int takeInOperONE;
        //int takeInOperTWO;
        checker = stackObject1.peekPlace(index+1);
        checked = IsOperator(checker);
        //cout << IsOperator(checker) << endl;
        if(checked == 1) {
            stackObject3.push(checker);
            //takeInOperONE = stackObject2.pop();
            //takeInOperTWO = stackObject2.pop();
            //cout << takeInOperONE << endl;
            //cout << takeInOperTWO << endl;
            //Operate(checker,)
            cout << stackObject3.pop() << endl;
        }
        //cout << stackObject4.peekPlace(index+1) << endl;

    }

    //cout << stackObject3.pop() << endl;

    return 0;
}
```

3.23

```
#include <iostream>

using namespace std;

#define SIZE 10

    struct stack {
      stack();
      void push(char ch);
      char pop();
      int isempty();
      int peekPlace(int n);
    private:
      char stackData[SIZE];
      int topOfStack;
    };
```

```
        stack::stack()
        {
         cout << "Constructing a stack\n";
         topOfStack = 0;
        }

        void stack::push(char ch)
        {
         if(topOfStack==SIZE) {
           cout << "Stack is full\n";
           return;
         }
         stackData[topOfStack] = ch;
         topOfStack++;
        }

        char stack::pop()
        {
         if(topOfStack==0) {
           cout << "Stack is empty\n";
           return 0; // return null on empty stack
         }
         topOfStack--;
         return stackData[topOfStack];
        }

        int stack::isempty()
        {
return (topOfStack==0?1:0);
        }

        int stack::peekPlace(int n)
        {
if(!isempty())
{
  //cout << "\nElement at top is " << stackData[topOfStack-n] << endl;
  return stackData[topOfStack-n];
}
else
{
  cout << "\nStack is empty";
  return 0;
}
        }
```

```
bool IsOperand(char C)
    {
        if(C >= '0' && C <= '9') return true;
        if(C >= 'a' && C <= 'z') return true;
        if(C >= 'A' && C <= 'Z') return true;
        return false;
    }

bool IsOperator(char C)
    {
        if(C == '+' || C == '-' || C == '*' || C == '/' || C== '$')
            return true;

        return false;
    }

//int Operate(int oper,) {


//  return 0;
//}

int main() {
    stack stackObject1, stackObject2, stackObject3, stackObject4;
    int index;

    stackObject1.push('1');
    stackObject1.push('+');
    stackObject1.push('2');
    stackObject1.push('*');
    stackObject1.push('4');
    stackObject1.push('-');
    stackObject1.push('3');


    cout << "Operands? (stack 2)" << endl;
    for(index = 0; index < 7; index++) {
        int checker;
        int checked;
        checker = stackObject1.peekPlace(index+1);
        checked = IsOperand(checker);
        //cout << checker << endl;
        if(checked == 1) {
            stackObject2.push(checker);
            stackObject4.push(checker);
            //cout << "HEY" << endl;
            //cout << stackObject4.pop() << endl;
            cout << stackObject2.pop() << endl;
        }
        //cout << IsOperand(checker) << endl;
    }


    cout << "Operators? (stack 3)" << endl;
    for(index = 0; index < 7; index++) {
        int checker;
        int checked;
        //int takeInOperONE;
        //int takeInOperTWO;
        checker = stackObject1.peekPlace(index+1);
        checked = IsOperator(checker);
        //cout << IsOperator(checker) << endl;
        if(checked == 1) {
            stackObject3.push(checker);
            //takeInOperONE = stackObject2.pop();
            //takeInOperTWO = stackObject2.pop();
            //cout << takeInOperONE << endl;
            //cout << takeInOperTWO << endl;
            //Operate(checker,)
            cout << stackObject3.pop() << endl;
        }
        //cout << stackObject4.peekPlace(index+1) << endl;

    }

    //cout << stackObject3.pop() << endl;

    return 0;
}
```