```
/* ***************************************************
 * Assignment : 3 - Stacks Queue Deque
 * Name       : Lina Kang
 * Student ID : 1072568
 * CS1D       : MW 2:30 - 5:00
 * Due Date   : 09/09/20
 * ***************************************************
 * -------------------DESCRIPTION-----------------------
 *
 * This assignment involves different practices of
 * stacks, queues, and deques, both STL and non-STL,
 * using push and pop methods
 *
 * ***************************************************
 * ----------------------OUTPUT-------------------------

***************************************************
* Assignment : 3 - Stacks Queue Deque
* Name       : Lina Kang
* Student ID : 1072568
* CS1D       : MW 2:30 - 5:00
* Due Date   : 09/09/20
***************************************************
This assignment involves different practices of
stacks, queues, and deques, both STL and non-STL,
using push and pop methods

------------------------------
Part A: Print Stack Top to Bottom
String Stack   Double Stack
------------------------------
Don            88.64
Mark           2019.1
JoAnn          123.123
Eric           200.12
Jordyn         888.55
Jennifer

------------------------------
Part B: Deletion of "Jordyn" and "123.123"
String Stack   Double Stack
------------------------------
Jennifer       200.12
               888.55

------------------------------
Part C: Implement a linked list stack
String Stack   Double Stack
------------------------------
Don            88.64
Mark           2019.1
JoAnn          123.123
Eric           200.12
Jordyn         888.55
Jennifer

------------------------------
```

```
Part C: Deletion of "Jordyn" and "123.123"
String Stack   Double Stack
------------------------------
Jennifer       200.12
               888.55

------------------------------
Part E: Implement Queue
String Queue   Double Queue
------------------------------
Jennifer       888.55
Jordyn         200.12
Eric           123.123
JoAnn          2019.1
Mark           88.64
Don

------------------------------
Part F: Deletion of "JoAnn" and "123.123"
String Queue   Double Queue
------------------------------
Mark           2019.1
Don            88.64

------------------------------
Part G: Implement Deque
String Deque   Double Deque
------------------------------
Jennifer       888.55
Jordyn         200.12
Eric           123.123
JoAnn          2019.1
Mark           88.64
Don

------------------------------
Part H: Deletion of "JoAnn" and "200.12"
String Deque   Double Deque
------------------------------
Jennifer       123.123
Jordyn         2019.1
               88.64

------------------------------
Part I: Parentheses Match Check
------------------------------
True: {2x+5}(6x+4)
True: (12x+6){2x-4}
True: (2x+7)(12x+6)
False: {{8x+5)-5x[9x+3]})
False: (((4x+8)-x[4x+3])))
True: [(5x-5)-4x[6x+2]]

 * ************************************************/

#include <iomanip>
#include <stack>
```

```cpp
#include "linkedStack.h"
#include "linkedQueue.h"
#include "linkedDeque.h"

//print Stack
void print(stack<string> str, stack<double> dob, string partDescription)
{
    cout << left;
    cout << "------------------------------" << endl <<
            "Part " << partDescription << endl;
    cout << setw(15) << "String Stack" <<
            setw(15) << "Double Stack" << endl <<
            "------------------------------" << endl;

    // Determine how many lines should be printed
    int loop = 6;
    if(dob.size() > str.size()) loop = dob.size();
    else loop = str.size();

    // Print whatever is remaining within the containers
    for(int i = 0; i < loop; i++)
    {
        if(!str.empty())
        {
            cout << setw(15) << str.top();
            str.pop();
        }
        else // prints an empty column if container is emptied out
        {
            cout << setw(15) << " ";
        }
        if(!dob.empty())
        {
            cout << setw(15) << dob.top();
            dob.pop();
        }
        else // prints an empty column if container is emptied out
        {
            cout << setw(15) << " ";
        }
        cout << endl;
    }
    cout << endl;
}

//print LinkedStackType
void print(linkedStackType<string> str, linkedStackType<double> dob, string partDescription)
{
    cout << left;
    cout << "------------------------------" << endl <<
            "Part " << partDescription << endl;
    cout << setw(15) << "String Stack" <<
            setw(15) << "Double Stack" << endl <<
            "------------------------------" << endl;

    int loop = 6;
    if(dob.getSize() > str.getSize()) loop = dob.getSize();
```

```
172     else loop = str.getSize();
173
174     for(int i = 0; i < loop; i++)
175     {
176         if(!str.isEmptyStack())
177         {
178             cout << setw(15) << str.getTop();
179             str.pop();
180         }
181         else
182         {
183             cout << setw(15) << " ";
184         }
185
186         if(!dob.isEmptyStack())
187         {
188             cout << setw(15) << dob.getTop();
189             dob.pop();
190         }
191         else
192         {
193             cout << setw(15) << " ";
194         }
195
196         cout << endl;
197     }
198
199     cout << endl;
200 }
201
202 //print QueueType
203 void print(linkedQueueType<string> str, linkedQueueType<double> dob, string partDescription)
204 {
205     cout << left;
206     cout << "------------------------------" << endl <<
207             "Part " << partDescription << endl;
208     cout << setw(15) << "String Queue" <<
209             setw(15) << "Double Queue" << endl <<
210             "------------------------------" << endl;
211
212     nodeType<string> *currentStr = str.getFront();
213     nodeType<double> *currentDob = dob.getFront();
214
215     int loop = 6;
216     if(dob.getSize() > str.getSize()) loop = dob.getSize();
217     else loop = str.getSize();
218
219     for(int i = 0; i < loop; i++)
220     {
221         if(currentStr != NULL)
222         {
223             cout << setw(15) << currentStr->info;
224             currentStr = currentStr->link;
225         }
226         else
227         {
228             cout << setw(15) << " ";
```

```
229         }
230
231         if(currentDob != NULL)
232         {
233             cout << setw(15) << currentDob->info;
234             currentDob = currentDob->link;
235         }
236         else
237         {
238             cout << setw(15) << " ";
239         }
240
241         cout << endl;
242     }
243
244     cout << endl;
245 }
246
247 //print Deque
248 void print(linkedDequeType<string> str, linkedDequeType<double> dob, string partDescription)
249 {
250     cout << left;
251     cout << "------------------------------" << endl <<
252             "Part " << partDescription << endl;
253     cout << setw(15) << "String Deque" <<
254             setw(15) << "Double Deque" << endl <<
255             "------------------------------" << endl;
256
257     nodeDeque<string> *currentStr = str.getTop();
258     nodeDeque<double> *currentDob = dob.getTop();
259
260     int loop = 6;
261     if(dob.getSize() > str.getSize()) loop = dob.getSize();
262     else loop = str.getSize();
263
264     for(int i = 0; i < loop; i++)
265     {
266         if(currentStr != NULL)
267         {
268             cout << setw(15) << currentStr->info;
269             currentStr = currentStr->back;
270         }
271         else
272         {
273             cout << setw(15) << " ";
274         }
275
276         if(currentDob != NULL)
277         {
278             cout << setw(15) << currentDob->info;
279             currentDob = currentDob->back;
280         }
281         else
282         {
283             cout << setw(15) << " ";
284         }
285
```

```
286         cout << endl;
287     }
288 
289     cout << endl;
290 }
291 
292 // Checks whether parentheses match on an equation using linkedStackType
293 bool ParenMatch(string X, int n)
294 {
295     linkedStackType<char> S;
296 
297     //uses switch statements to cover 3 different types of grouping symbols
298     for(int i = 0; i < n; i++)
299     {
300         switch(X[i])
301         {
302         case '(':
303             S.push(X[i]);
304             break;
305         case '{':
306             S.push(X[i]);
307             break;
308         case '[':
309             S.push(X[i]);
310             break;
311         case ')':
312             if(S.isEmptyStack())
313                 return false;
314             else if(S.getTop() == '(')
315                 S.pop();
316             else
317                 return false;
318             break;
319         case '}':
320             if(S.isEmptyStack())
321                 return false;
322             else if(S.getTop() == '{')
323                 S.pop();
324             else
325                 return false;
326             break;
327         case ']':
328             if(S.isEmptyStack())
329                 return false;
330             else if(S.getTop() == '[')
331                 S.pop();
332             else
333                 return false;
334             break;
335         default:
336             break;
337         }
338     }
339     return true;
340 }
341 int main()
342 {
```

```
343     cout << "*******************************************************\n"
344             "* Assignment : 3 - Stacks Queue Deque \n"
345             "* Name       : Lina Kang \n"
346             "* Student ID : 1072568 \n"
347             "* CS1D       : MW 2:30 - 5:00 \n"
348             "* Due Date   : 09/09/20 \n"
349             "*******************************************************\n"
350             "This assignment involves different practices of \n"
351             "stacks, queues, and deques, both STL and non-STL, \n"
352             "using push and pop methods  \n\n";
353
354     // A - Implement a Stack
355
356     stack <string> stackString;
357     stack <double> stackDouble;
358
359     stackString.push("Jennifer");
360     stackString.push("Jordyn");
361     stackString.push("Eric");
362     stackString.push("JoAnn");
363     stackString.push("Mark");
364     stackString.push("Don");
365
366     stackDouble.push(888.55);
367     stackDouble.push(200.12);
368     stackDouble.push(123.123);
369     stackDouble.push(2019.1);
370     stackDouble.push(88.64);
371
372     print(stackString, stackDouble,
373           "A: Print Stack Top to Bottom");
374
375     // B - Perform Deletion from a stack
376
377     stackString.pop();
378     stackString.pop();
379     stackString.pop();
380     stackString.pop();
381     stackString.pop();
382
383     stackDouble.pop();
384     stackDouble.pop();
385     stackDouble.pop();
386
387     print(stackString, stackDouble,
388           "B: Deletion of \"Jordyn\" and \"123.123\"");
389
390     // C - Implement a non-STL stack
391
392     linkedStackType<string> linkedString;
393     linkedStackType<double> linkedDouble;
394
395     linkedString.push("Jennifer");
396     linkedString.push("Jordyn");
397     linkedString.push("Eric");
398     linkedString.push("JoAnn");
399     linkedString.push("Mark");
```

```
400     linkedString.push("Don");
401
402     linkedDouble.push(888.55);
403     linkedDouble.push(200.12);
404     linkedDouble.push(123.123);
405     linkedDouble.push(2019.1);
406     linkedDouble.push(88.64);
407
408     print(linkedString, linkedDouble,
409           "C: Implement a linked list stack");
410
411     // D - Perform Deletion from non-STL stack
412
413     linkedString.pop();
414     linkedString.pop();
415     linkedString.pop();
416     linkedString.pop();
417     linkedString.pop();
418
419     linkedDouble.pop();
420     linkedDouble.pop();
421     linkedDouble.pop();
422
423     print(linkedString, linkedDouble,
424           "C: Deletion of \"Jordyn\" and \"123.123\"");
425
426     // E - Implement a non-STL queue
427
428     linkedQueueType<string> queueString;
429     linkedQueueType<double> queueDouble;
430
431     queueString.addQueue("Jennifer");
432     queueString.addQueue("Jordyn");
433     queueString.addQueue("Eric");
434     queueString.addQueue("JoAnn");
435     queueString.addQueue("Mark");
436     queueString.addQueue("Don");
437
438     queueDouble.addQueue(888.55);
439     queueDouble.addQueue(200.12);
440     queueDouble.addQueue(123.123);
441     queueDouble.addQueue(2019.1);
442     queueDouble.addQueue(88.64);
443
444     print(queueString, queueDouble, "E: Implement Queue");
445
446     // F - Perform Deletion of queue
447
448     queueString.deQueue();
449     queueString.deQueue();
450     queueString.deQueue();
451     queueString.deQueue();
452
453     queueDouble.deQueue();
454     queueDouble.deQueue();
455     queueDouble.deQueue();
456
```

```
457     print(queueString, queueDouble,
458           "F: Deletion of \"JoAnn\" and \"123.123\"");
459
460     // G - Implement a non-STL deque
461
462     linkedDequeType<string> dequeString;
463     linkedDequeType<double> dequeDouble;
464
465     dequeString.pushTop("Jennifer");
466     dequeString.pushTop("Jordyn");
467     dequeString.pushTop("Eric");
468     dequeString.pushTop("JoAnn");
469     dequeString.pushTop("Mark");
470     dequeString.pushTop("Don");
471
472     dequeDouble.pushTop(888.55);
473     dequeDouble.pushTop(200.12);
474     dequeDouble.pushTop(123.123);
475     dequeDouble.pushTop(2019.1);
476     dequeDouble.pushTop(88.64);
477
478     print(dequeString, dequeDouble, "G: Implement Deque");
479
480     // H - Perform Deletion from deque
481
482     dequeString.popTop();
483     dequeString.popTop();
484     dequeString.popTop();
485     dequeString.popTop();
486
487     dequeDouble.popBottom();
488     dequeDouble.popBottom();
489
490     print(dequeString, dequeDouble,
491           "H: Deletion of \"JoAnn\" and \"200.12\"");
492
493     // I - Partheneses Match Algorithm
494
495     cout << "--------------------------------"  << endl <<
496             "Part I: Parentheses Match Check" << endl <<
497             "--------------------------------"  << endl;
498
499     string a = "{2x+5}(6x+4)";
500     string b = "(12x+6){2x-4}";
501     string c = "(2x+7)(12x+6)";
502     string d = "{{8x+5)-5x[9x+3]})";
503     string e = "(((4x+8)-x[4x+3])))";
504     string f = "[(5x-5)-4x[6x+2]]";
505
506     if(ParenMatch(a, a.length()))
507         cout << "True: " << a << endl;
508     else
509         cout << "False: " << a << endl;
510
511     if(ParenMatch(b, b.length()))
512         cout << "True: " << b << endl;
513     else
```

```
514         cout << "False: " << b << endl;
515
516     if(ParenMatch(c, c.length()))
517         cout << "True: " << c << endl;
518     else
519         cout << "False: " << c << endl;
520
521     if(ParenMatch(d, d.length()))
522         cout << "True: " << d << endl;
523     else
524         cout << "False: " << d << endl;
525
526     if(ParenMatch(e, e.length()))
527         cout << "True: " << e << endl;
528     else
529         cout << "False: " << e << endl;
530
531     if(ParenMatch(f, f.length()))
532         cout << "True: " << f << endl;
533     else
534         cout << "False: " << f << endl;
535
536 }
537
538 // Header files - linkedStack.h, linkedQueue.h, linkedDeque.h
539
540 #ifndef LINKEDSTACK_H_
541 #define LINKEDSTACK_H_
542
543 #include <iostream>
544 using namespace std;
545
546 template <class Type>
547 struct nodeType
548 {
549     Type info;
550     nodeType<Type> *link;
551 };
552 template <class Type>
553 class linkedStackType
554 {
555 public:
556     bool isEmptyStack();
557     bool isFullStack();
558     Type getTop();
559     void push(const Type&newItem);
560     void pop(Type & poppedElement);
561     linkedStackType();
562     linkedStackType(const linkedStackType<Type>& otherStack);
563     ~linkedStackType();
564
565     void pop();
566     int getSize() const;
567
568 private:
569     nodeType<Type> *top; //pointer to the stack};
570     int size;
```

```
571 };
572
573 template <class Type>
574 bool linkedStackType<Type>::isEmptyStack()
575 {
576     return(top==NULL);
577 }
578
579 template <class Type>
580 bool linkedStackType<Type>::isFullStack()
581 {
582     return 0;
583 }
584
585 template <class Type>
586 Type linkedStackType<Type>::getTop()
587 {
588     return top->info;
589 }
590
591 template<class Type>
592 void linkedStackType<Type>::push(const Type& newElement)
593 {
594     nodeType<Type> *newNode;
595
596     newNode = new nodeType<Type>;
597     newNode->info = newElement;
598     newNode->link = top;
599     top = newNode;
600
601     size++;
602 }
603
604 template <class Type>
605 void linkedStackType<Type>::pop()
606 {
607     nodeType <Type> *temp;
608
609     temp = top;
610     top = top->link;
611     delete temp;
612     size--;
613 }
614 template <class Type>
615 void linkedStackType<Type>::pop(Type& poppedElement)
616 {
617     nodeType <Type> *temp;
618     poppedElement = top->info;
619
620     temp = top;
621     top = top->link;
622     delete temp;
623     size--;
624 }
625 template<class Type>
626 linkedStackType<Type>::linkedStackType()
627 {
```

```
628     top = NULL;
629     size = 0;
630 }
631
632 template<class Type>
633 linkedStackType<Type>::linkedStackType(const linkedStackType<Type>& otherStack)
634 {
635     nodeType<Type> *newNode, *current, *last;
636
637     if(otherStack.top == NULL)
638         top = NULL;
639     else
640     {
641         current = otherStack.top;
642
643         top = new nodeType<Type>;
644         top->info = current->info;
645         top->link = NULL;
646
647         last = top;
648
649         current = current->link;
650
651         while(current != NULL)
652         {
653             newNode = new nodeType<Type>;
654             newNode->info = current->info;
655             newNode->link = NULL;
656             last->link = newNode;
657             last = newNode;
658             current = current->link;
659         }
660
661     }
662     size = otherStack.getSize();
663
664 }
665
666 template <class Type>
667 linkedStackType<Type>::~linkedStackType()
668 {
669     nodeType<Type> *temp;
670
671     while(top != NULL)
672     {
673         temp = top;
674         top = top->link;
675         delete temp;
676     }
677 }
678 template <class Type>
679 int linkedStackType<Type>::getSize() const
680 {
681     return size;
682 }
683
684 #endif /* LINKEDSTACK_H_ */
```

```
685
686 #ifndef LINKEDQUEUE_H_
687 #define LINKEDQUEUE_H_
688
689 #include <iostream>
690 using namespace std;
691
692 template <class Type>
693 class linkedQueueType
694 {
695 public:
696     bool isEmptyQueue();
697     void addQueue(const Type &newElement);
698     void deQueue();
699     linkedQueueType();
700     linkedQueueType(const linkedQueueType<Type> &otherQueue);
701     ~linkedQueueType();
702
703     nodeType<Type>* getFront();
704     int getSize() const;
705 private:
706     nodeType<Type> *front;
707     nodeType<Type> *rear;
708     int size;
709 };
710
711 template <class Type>
712 bool linkedQueueType<Type>::isEmptyQueue()
713 {
714     return (front == NULL);
715 }
716 template <class Type>
717 void linkedQueueType<Type>::addQueue(const Type &newElement)
718 {
719     nodeType<Type> *newNode;
720     newNode = new nodeType<Type>;
721     newNode->info = newElement;
722     newNode->link = NULL;
723     if (front == NULL)
724     {
725         front = newNode;
726         rear = newNode;
727     }
728     else
729     {
730         rear->link = newNode;
731         rear = rear->link;
732     }
733     size++;
734 }
735
736 template <class Type>
737 void linkedQueueType<Type>::deQueue()
738 {
739     nodeType<Type> *temp;
740     temp = front;
741     front = front->link;
```

```
742     delete temp;
743     if (front == NULL)
744         rear = NULL;
745     size--;
746 }
747 template <class Type>
748 linkedQueueType<Type>::linkedQueueType()
749 {
750     front = NULL;
751     rear = front;
752     size = 0;
753 }
754 template <class Type>
755 linkedQueueType<Type>::linkedQueueType(const linkedQueueType<Type> &otherQueue)
756 {
757     front = NULL;
758     rear = NULL;
759     size = 0;
760
761     nodeType<Type> *current;
762     current = otherQueue.front;
763     while(current != NULL)
764     {
765         addQueue(current->info);
766         current = current->link;
767     }
768 }
769 template <class Type>
770 linkedQueueType<Type>::~linkedQueueType()
771 {
772     nodeType<Type> *temp;
773     while (front != NULL)
774     {
775         temp = front;
776         front = front->link;
777         delete temp;
778     }
779     rear = NULL;
780 }
781 template <class Type>
782 nodeType<Type>* linkedQueueType<Type>::getFront()
783 {
784     return front;
785 }
786 template <class Type>
787 int linkedQueueType<Type>::getSize() const
788 {
789     return size;
790 }
791 #endif /* LINKEDQUEUE_H_ */
792
793 #ifndef LINKEDDEQUE_H_
794 #define LINKEDDEQUE_H_
795
796 template <class Type>
797 struct nodeDeque
798 {
```

```
799     Type info;
800     nodeDeque<Type> *front;
801     nodeDeque<Type> *back;
802 };
803
804 template <class Type>
805 class linkedDequeType
806 {
807 public:
808     bool isEmptyDeque();
809     void pushTop(Type);
810     void pushBottom(Type);
811     void popTop();
812     void popBottom();
813     linkedDequeType();
814     linkedDequeType(const linkedDequeType<Type> &);
815     ~linkedDequeType();
816
817     nodeDeque<Type>* getTop();
818     nodeDeque<Type>* getBottom();
819     int getSize() const;
820
821 private:
822     nodeDeque<Type> *top;
823     nodeDeque<Type> *bottom;
824     nodeDeque<Type> *temp;
825     int size;
826 };
827 template <class Type>
828 linkedDequeType<Type>::linkedDequeType()
829 {
830     top = NULL;
831     bottom = NULL;
832     temp = NULL;
833     size = 0;
834 }
835 template <class Type>
836 linkedDequeType<Type>::linkedDequeType(const linkedDequeType<Type>& otherDeque)
837 {
838     top = NULL;
839     bottom = NULL;
840     size = 0;
841
842     temp = otherDeque.top;
843     while(temp != NULL)
844     {
845         pushTop(temp->info);
846         temp = temp->back;
847     }
848 }
849 template <class Type>
850 linkedDequeType<Type>::~linkedDequeType()
851 {
852     while(top != NULL)
853     {
854         temp = top;
855         top = top->back;
```

```
856         delete temp;
857     }
858 }
859 template <class Type>
860 bool linkedDequeType<Type>::isEmptyDeque()
861 {
862     if(top == NULL && bottom == NULL)
863         return true;
864     return false;
865 }
866 template <class Type>
867 void linkedDequeType<Type>::pushTop(Type item)
868 {
869     nodeDeque<Type> *newNode = new nodeDeque<Type>;
870     newNode->info = item;
871     if(top != NULL)
872     {
873         top->front = newNode;
874         newNode->back = top;
875         newNode->front = NULL;
876         top = newNode;
877     }
878     else
879     {
880         top = newNode;
881         bottom = newNode;
882         top->front = NULL;
883         top->back = bottom;
884         bottom->front = newNode;
885         bottom->back = NULL;
886     }
887     size++;
888 }
889 template <class Type>
890 void linkedDequeType<Type>::pushBottom(Type item)
891 {
892     nodeDeque<Type> *newNode = new nodeDeque<Type>;
893     newNode->info = item;
894     if(top != NULL)
895     {
896         bottom->back = newNode;
897         newNode->front = bottom;
898         newNode->back = NULL;
899         bottom = newNode;
900     }
901     else
902     {
903         top = newNode;
904         bottom = newNode;
905         top->front = NULL;
906         top->back = bottom;
907         bottom->front = newNode;
908         bottom->back = NULL;
909     }
910     size++;
911 }
912 template <class Type>
```

```
913 void linkedDequeType<Type>::popTop()
914 {
915     temp = top;
916     top = top->back;
917     top->front = NULL;
918     delete temp;
919     size--;
920 }
921 template <class Type>
922 void linkedDequeType<Type>::popBottom()
923 {
924     temp = bottom;
925     bottom = bottom->front;
926     bottom->back = NULL;
927     delete temp;
928     size--;
929 }
930 template <class Type>
931 nodeDeque<Type>* linkedDequeType<Type>::getTop()
932 {
933     return top;
934 }
935 template <class Type>
936 nodeDeque<Type>* linkedDequeType<Type>::getBottom()
937 {
938     return bottom;
939 }
940 template <class Type>
941 int linkedDequeType<Type>::getSize() const
942 {
943     return size;
944 }
945
946 #endif /* LINKEDDEQUE_H_ */
947
```