1. #pragma once
2. template<typename T>
3. class MyArray
4. {
5. public:
6. MyArray();
7. MyArray(int len);
8. int getLength() const;
9. bool isFull() const;
10. void Insert(int pos,const T& P);
11. void Delete(int pos);
12. int getLocation(const T& P) const;
13. void Reverse();
14. void interValue(const T& Min, const T& Max);
15. void Print();
16. ~MyArray();
17. private:
18. int \_length;
19. T\* \_Arr;
20. int \_MAX\_LENGTH;
21. };
22. #include "myarray.h"
23. #include <iostream>
24. template<typename T>
25. MyArray<T>::MyArray() : \_MAX\_LENGTH(1000) {
26. \_Arr = new T[\_MAX\_LENGTH];
27. \_length = 0;
28. }
29. template<typename T>
30. MyArray<T>::MyArray(int len) : \_MAX\_LENGTH(len) {
31. \_Arr = new T[\_MAX\_LENGTH];
32. \_length = 0;
33. }
34. template<typename T>
35. int MyArray<T>::getLength() const {
36. return \_length;
37. }
38. template<typename T>
39. bool MyArray<T>::isFull() const {
40. return \_length == \_MAX\_LENGTH - 1;
41. }
42. template<typename T>
43. void MyArray<T>::Insert(int pos, const T& P) {
44. if (isFull()) {
45. std::cout << "Now Array is Full" << std::endl;
46. return;
47. }
48. if (pos >= \_length) {
49. \_Arr[ \_length++] = P;
50. }
51. else {
52. pos--;
53. for (int i = \_length - 1; i >= pos; --i) {
54. \_Arr[i + 1] = \_Arr[i];
55. }
56. \_Arr[pos] = P;
57. \_length++;
58. }
59. }
60. template<typename T>
61. void MyArray<T>::Delete(int pos) {
62. pos--;
63. if (pos < 0 || pos >= \_MAX\_LENGTH) {
64. std::cout << "Incorrect location of deleted element" << std::endl;
65. return;
66. }
67. for (int i = pos; i < \_length; ++i) {
68. \_Arr[i] = \_Arr[i + 1];
69. }
70. \_length--;
71. }
72. template<typename T>
73. int MyArray<T>::getLocation(const T& P) const {
74. *// maybe mulit answer*
75. for (int i = 0; i < \_length; ++i) {
76. if (P == \_Arr[i]) return i + 1;
77. }
78. return 0;
79. }
80. template<typename T>
81. void MyArray<T>::Reverse() {
82. int Mid = \_length / 2;
83. for (int i = 0; i < Mid; ++i) {
84. T temp = \_Arr[i];
85. \_Arr[i] = \_Arr[\_length - 1 - i];
86. \_Arr[\_length - 1 - i] = temp;
87. }
88. }
89. template<typename T>
90. void MyArray<T>::interValue(const T& Min, const T& Max) {
91. T\* preArr = \_Arr;
92. T\* sifArr = new T[\_MAX\_LENGTH];
93. int cnt = 0;
94. for (int i = 0; i < \_length; ++i) {
95. if (\_Arr[i] >= Min && \_Arr[i] <= Max) {
96. sifArr[cnt++] = \_Arr[i];
97. }
98. }
99. delete[] preArr;
100. \_Arr = sifArr;
101. \_length = cnt;
102. }
103. template<typename T>
104. void MyArray<T>::Print() {
105. for (int i = 0; i < \_length; ++i) {
106. std::cout << \_Arr[i] << " ";
107. }
108. std::cout << "\n";
109. }
110. template<typename T>
111. MyArray<T>::~MyArray() {
112. delete[] \_Arr;
113. }