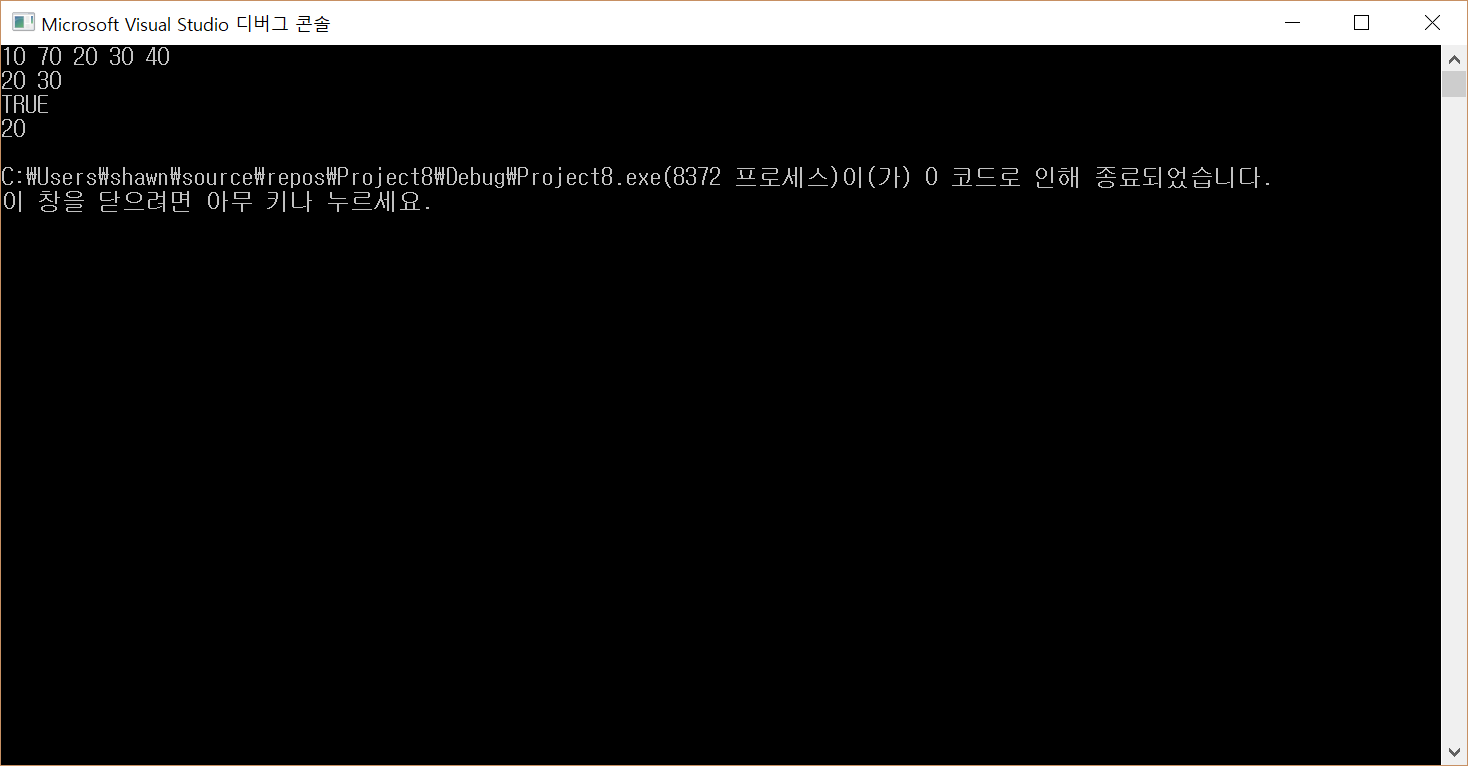
1. 코드 및 주석
2. #include <stdio.h>
3. #include <malloc.h>
4. #include <stdlib.h>
5. #define TRUE 1
6. #define FALSE 0
7. typedef int element;
8. typedef struct ListNode {//노드 구조체 선언
9. element data;
10. struct ListNode \*link;
11. } ListNode;
12. typedef struct {
13. ListNode \*head;// Head pointer
14. ListNode \*tail;// Tail pointer
15. int length;// # of nodes
16. } ListType;
17. void init(ListType \*list) {//리스트 초기화 함수
18. list->head = NULL;
19. list->tail = NULL;
20. list->length = 0;
21. }
22. int is\_empty(ListType \*list) {
23. if (list->head == NULL)//헤드포인터가 비어있으면
24. return 1;
25. else
26. return 0;
27. }//empty인지 알아보는 함수
28. int get\_length(ListType\* list)//길이 리턴 함수
29. {
30. return list->length;
31. }
32. ListNode \*get\_node\_at(ListType \*list, int pos) {
33. int i;
34. ListNode \*tmp\_node = list->head;//tmp\_node가 일단 헤드포인터를 가르키도록 한다.
35. if (pos < 0)
36. return NULL;//패러미터로 받은 인덱스가 0보다 작을 경우
37. if (pos > list->length)
38. return NULL;
39. for (i = 0; i < pos; i++) {//원하는 노드에 접근하기 위한 과정
40. tmp\_node = tmp\_node->link;
41. }
42. return tmp\_node;
43. }
44. void add(ListType \*list, int position, element data) {//Insert new data at the 'position'
45. ListNode \*p;//임시로 노드 포인터를 담을 것.
46. p = (ListNode\*)malloc(sizeof(ListNode));
47. p->data = data;
48. int i;
49. ListNode \*ori\_head = list->head;
51. if (position == 1) {
52. p->link = ori\_head;
53. list->head = p;
54. return;
55. }
56. for (int i = 1; i < position - 1; i++) {
57. ori\_head = ori\_head->link;
58. }
59. p->link = ori\_head->link;
60. ori\_head->link = p;
61. }
62. void delete(ListType \*list, int pos) {
63. int i;
64. if (list->head == NULL)
65. return;
66. ListNode \*p = list->head;
67. if(pos==0){
68. list->head = p->link;//맨 앞에것 지우면 뒤의 노드를 헤드로
69. free(p);//제거
70. return;
71. }
73. else {
74. for (i = 1; i < pos - 1; i++) {
75. p = p->link;
76. }
77. }
78. ListNode \*next\_p = p->link->link;
79. free(p->link);
80. p->link = next\_p;
81. }
82. element get\_entry(ListType \*list, int pos) {
83. ListNode \*p=list->head;
85. int i;
86. if (pos == 0)
87. return(p->data);
88. else {
89. for (i = 0; i < pos; i++) {
90. p = p->link;
91. }
92. return (p->data);
93. }
94. }
95. void display(ListType \*list) {
96. ListNode \*p = list->head;
97. while (p != NULL) {
98. printf("%d ", p->data);
99. p = p->link;
100. }
101. }
102. int is\_in\_list(ListType \*list, element item) {
103. ListNode \*p;
104. p = list->head;
105. while ((p != NULL)) {
106. if (p->data == item)
107. break;
108. p = p->link;
109. }
110. if (p == NULL)
111. return TRUE;
112. else
113. return FALSE;
114. }
115. void add\_first(ListType \*list, element item) {
116. ListNode \*newNode;//새로운 노드
117. newNode = (ListNode\*)malloc(sizeof(ListNode));
118. newNode->data = item;
120. if (list->head == NULL) {//만약에 아무것도 없는 리스트라면
121. list->head = newNode;//head에 새로운 노드값 넣음
122. newNode->link = NULL;
123. list->tail = newNode;//꼬리도 같은 주소 저장
124. }
125. else {
126. newNode->link = list->head;
127. list->head = newNode;
128. }
129. (list->length)++;//리스트 길이 값 증가
130. }
131. void add\_last(ListType \*list, element item) {
132. ListNode \*newNode;
133. newNode = (ListNode\*)malloc(sizeof(ListNode));
134. newNode->data = item;
135. newNode->link = NULL;//새로운 노드의 link에 null 넣음(마지막 노드니까)
136. list->tail->link = newNode;//기존의 tail주소가 가르키는 마지막 노드 link에 새로운 노드 주소 입력
137. list->tail = newNode;//이제 tail은 새로운 노드를 가리킨다.
138. (list->length)++;//리스트 길이 값 증가
139. }
140. void delete\_first(ListType \*list) {
141. ListNode \*removed = list->head;
142. list->head = list->head->link;//헤드포인터를 첫번째 노드의 link의 주소로 대체
143. free(removed);
144. (list->length)--;//리스트 길이 감소
145. }
146. void delete\_last(ListType \*list) {//맨 마지막의 노드를 지운다.
147. ListNode \*removed = list->tail;
148. ListNode \*p = list->head;;//tail전의 노드 주소받기
149. while (p->link != list->tail) {//link가 tail이면 tail전의 노드이다!!!!!
150. p = p->link;
151. }
152. list->tail = p;//tail에 제거될 전의 노드 주소 저장!!!!
153. free(removed);
154. p->link = NULL;
155. (list->length)--;
156. }
157. int main(void)
158. {
159. ListType list1;
160. init(&list1);
161. add\_first(&list1, 20);
162. add\_last(&list1, 30);
163. add\_first(&list1, 10);
164. add\_last(&list1, 40);
165. add(&list1, 2, 70);
166. display(&list1);
167. printf("\n");
168. delete(&list1, 2);
169. delete\_first(&list1);
170. delete\_last(&list1);
171. display(&list1);
172. printf("\n%s\n", is\_in\_list(&list1, 20) == 0 ? "TRUE" : "FALSE");
173. printf("%d\n", get\_entry(&list1, 0));
174. return 0;
175. }

2. 결과 콘솔 창



3. 코드 분석 및 설명

1) 원하는 포지션에 접근하기 위한 가장 중요한 알고리즘

1. for (i = 0; i < pos; i++) {//원하는 노드에 접근하기 위한 과정
2. tmp\_node = tmp\_node->link;
3. }

2) first에 노드를 집어넣거나, 빼기 위한 과정에서 가장 중요한 코드

1. else {
2. newNode->link = list->head;
3. list->head = newNode;
4. }

list->head = list->head->link;//헤드포인터를 첫번째 노드의 link의 주소로 대체

3) last 에 노드를 집어넣거나, 빼기 위한 과정에서 가장 중요한 코드

1. list->tail->link = newNode;//기존의 tail주소가 가르키는 마지막 노드 link에 새로운 노드 주소 입력
2. list->tail = newNode;//이제 tail은 새로운 노드를 가리킨다.
3. while (p->link != list->tail) {//link가 tail이면 tail전의 노드이다!!!!!
4. p = p->link;
5. }
6. list->tail = p;//tail에 제거될 전의 노드 주소 저장!!!!