|  |  |  |  |
| --- | --- | --- | --- |
| 2022/2 『자료구조』실습 보고서 | | | |
| 제목 | 6장 실습( O ) 과제( ) | 제출일자 | 2022.  10 .    20 . |
| 학번 | 201911608 | 이름 | 김지환 |

|  |
| --- |
| 1. 프로그램 6.1  #include <stdio.h>  #include <stdlib.h>  #define MAX\_LIST\_SIZE 100  typedef int element;  typedef struct {  element array[MAX\_LIST\_SIZE];  int size;  } ArrayListType;  void error(char\* message) {  fprintf(stderr, "%s\n", message);  exit(1);  }  void init(ArrayListType\* L) {  L->size = 0;  }  int is\_empty(ArrayListType\* L) {  return L->size == 0;  }  int is\_full(ArrayListType\* L) {  return L->size == MAX\_LIST\_SIZE;  }  element get\_entry(ArrayListType\* L, int pos) {  if (pos < 0 || pos >= L->size)  error("위치 오류");  return L->array[pos];  }  void print\_list(ArrayListType\* L) {  int i;  for (i = 0; i < L->size; i++)  printf("%d->", L->array[i]);  printf("\n");  }  void insert\_last(ArrayListType\* L, element item) {  if (L->size >= MAX\_LIST\_SIZE)  error("리스트 오버플로우");  L->array[L->size++] = item;  }  void insert(ArrayListType\* L, int pos, element item) {  if (!is\_full(L) && (pos >= 0) && (pos <= L->size)) {  for (int i = (L->size - 1); i >= pos; i--)  L->array[i + 1] = L->array[i];  L->array[pos] = item;  L->size++;  }  }  element delete(ArrayListType\* L, int pos) {  element item;  if (pos < 0 || pos >= L->size)  error("위치 오류");  item = L->array[pos];  for (int i = pos; i < (L->size - 1); i++)  L->array[i] = L->array[i + 1];  L->size--;  return item;  }  int main(void) {    ArrayListType list;  init(&list);  insert(&list, 0, 10); print\_list(&list); // 0번째 위치에 10 추가  insert(&list, 0, 20); print\_list(&list); // 0번째 위치에 20 추가  insert(&list, 0, 30); print\_list(&list); // 0번째 위치에 30 추가  insert\_last(&list, 40); print\_list(&list); // 맨 끝에 40 추가  delete(&list, 0); print\_list(&list); // 0번째 항목 삭제  return 0;  }  실행결과    -> size를 0으로 초기화 후 리스트가 포화 상태이지 않으면 값을 삽입한다.  ㅇ 값을 삽입하는 과정  -> 삽입할 list와 index, value를 파라미터로 받아서 index~size까지 모든 원소를 한칸씩 뒤로 옮기고 해당 index에 값을 삽입한다.  ㅇ 값을 제거하는 과정  -> 제거할 list와 index를 파라미터로 받아서 현재 리스트의 size보다 큰 값이 아니면 삭제하는 과정이다.  ㅇ Insert\_last  -> size가 최대 사이즈가 되면 종료, 그렇지 않으면 리스트의 마지막값에 값을 삽입하고 size++ |
| 2. 배열 리스트의 QUIZ 01  본문에 나와 있지 않은 다음과 같은 리스트 ADT의 연산을 구현하여 보자.  \* clear(list)  \* repalce(list, pos, item)  \* get\_length(list)  배열 리스트 추가 ADT  void clear(ArrayListType\* L) {  if (L->size == 0) {  printf("현재 리스트가 비어 있습니다.");  }  for (int i = 0; i < L->size; i++) L->array[i] = 0;  L->size = 0;  }  void replace(ArrayListType\* L, int pos, element item) {  if (pos < 0 || pos >= L->size) error("위치 오류");  L->array[pos] = item;  }  int get\_length(ArrayListType\* L) {  return L->size;  }  전체 소스 코드  #include <stdio.h>  #include <stdlib.h>  #define MAX\_LIST\_SIZE 100  typedef int element;  typedef struct {  element array[MAX\_LIST\_SIZE];  int size;  } ArrayListType;  void error(char\* message) {  fprintf(stderr, "%s\n", message);  exit(1);  }  void init(ArrayListType\* L) {  L->size = 0;  }  int is\_empty(ArrayListType\* L) {  return L->size == 0;  }  int is\_full(ArrayListType\* L) {  return L->size == MAX\_LIST\_SIZE;  }  element get\_entry(ArrayListType\* L, int pos) {  if (pos < 0 || pos >= L->size)  error("위치 오류");  return L->array[pos];  }  void print\_list(ArrayListType\* L) {  int i;  printf("리스트 : ");  for (i = 0; i < L->size; i++)  printf("%d->", L->array[i]);  printf("\n");  }  void insert\_last(ArrayListType\* L, element item) {  if (L->size >= MAX\_LIST\_SIZE)  error("리스트 오버플로우");  L->array[L->size++] = item;  }  void insert(ArrayListType\* L, int pos, element item) {  if (!is\_full(L) && (pos >= 0) && (pos <= L->size)) {  for (int i = (L->size - 1); i >= pos; i--)  L->array[i + 1] = L->array[i];  L->array[pos] = item;  L->size++;  }  }  element delete(ArrayListType\* L, int pos) {  element item;  if (pos < 0 || pos >= L->size)  error("위치 오류");  item = L->array[pos];  for (int i = pos; i < (L->size - 1); i++)  L->array[i] = L->array[i + 1];  L->size--;  return item;  }  void clear(ArrayListType\* L) {  if (L->size == 0) {  printf("현재 리스트가 비어 있습니다.");  }  for (int i = 0; i < L->size; i++) L->array[i] = 0;  L->size = 0;  }  void replace(ArrayListType\* L, int pos, element item) {  if (pos < 0 || pos >= L->size) error("위치 오류");  L->array[pos] = item;  }  int get\_length(ArrayListType\* L) {  return L->size;  }  int main(void) {    ArrayListType list;  init(&list);  insert(&list, 0, 10); print\_list(&list);  insert(&list, 0, 20); print\_list(&list);  insert(&list, 0, 30); print\_list(&list);  insert\_last(&list, 40); print\_list(&list);  delete(&list, 0); print\_list(&list);  printf("===== replace(&list, 1, 100) =====\n");  replace(&list, 1, 100); print\_list(&list);  printf("List의 사이즈 = %d\n", get\_length(&list));  printf("===== clear(&list) =====\n");  clear(&list); print\_list(&list);    return 0;  }  실행결과 |
| 3. 프로그램 6.7#include <stdio.h>  #include <stdlib.h>  typedef int element;  typedef struct ListNode {  element data;  struct ListNode\* link;  } ListNode;  void error(char\* message) {  fprintf(stderr, "%s\n", message);  exit(1);  }  ListNode\* insert\_first(ListNode\* head, int value) {  ListNode\* p = (ListNode\*)malloc(sizeof(ListNode));  p->data = value;  p->link = head;  head = p;  return head;  }  ListNode\* insert(ListNode\* head, ListNode\* pre, element value) {  ListNode\* p = (ListNode\*)malloc(sizeof(ListNode));  p->data = value;  p->link = pre->link;  pre->link = p;  return head;  }  ListNode\* delete\_first(ListNode\* head) {  ListNode\* removed;  if (head == NULL) return NULL;  removed = head;  head = removed->link;  free(removed);  return head;  }  ListNode\* delete(ListNode\* head, ListNode\* pre) {  ListNode\* removed;  removed = pre->link;  pre->link = removed->link;  free(removed);  return head;  }  void print\_list(ListNode\* head) {  for (ListNode\* p = head; p != NULL; p = p->link)  printf("%d->", p->data);  printf("NULL \n");  }  int main(void) {  ListNode\* head = NULL;  for (int i = 0; i < 5; i++) {  head = insert\_first(head, i);  print\_list(head);  }  for (int i = 0; i < 5; i++) {  head = delete\_first(head);  print\_list(head);  }  return 0;  }  실행결과    ->  insert\_fisrt ADT 는 새로운 노드를 헤드와 연결하고 새로 연결된 노드를 반환하는 과정을 반복한다.  즉 기존의 head의 값이 새로 삽입되는 값의 다음을 나타내고 head를 새로 삽입되는 값으로 변경  delete\_first ADT는 head가 비어 있으면 NULL을 반환하고 그렇지 않으면 head를 head의 다음 값으로 변경 후 malloc으로 할당한 메모리를 반환 후 main 함수로 head를 반환한다.  print\_list ADRT는 head부터 NULL까지의 모든 data를 출력하며 head는 다음 값으로 변경되어 출력을 반복한다. |
| 4. 프로그램 6.7 QUIZ  01. 단순 연결 리스트에서 하나의 노드를 삭제하려면 어떤 노드를 가리키는 포인터 변수가 필요한가?  삭제할 노드의 이전 노드, 삭제할 노드가 필요하다.  A -> B -> C 에서 B 노드를 삭제 하려면 B의 이전 노드인 A를 가르키는 포인터 변수 pre와  삭제할 노드인 B 가르키는 포인터 변수 removed를 받아서 prev->link = removed->link를 통해서 A -> C로 B 노드를 삭제 할 수 있다.  02. 단순 연결 리스트에 존재하는 노드의 수를 계산하는 함수 get\_length()를 작성하라.  get\_length() ADT 구현  int get\_length(ListNode\* head) {  int size = 0;  for (ListNode\* p = head; p != NULL; p = p->link) size++;  return size;  }  전체 소스 코드  #include <stdio.h>  #include <stdlib.h>  typedef int element;  typedef struct ListNode {  element data;  struct ListNode\* link;  } ListNode;  void error(char\* message) {  fprintf(stderr, "%s\n", message);  exit(1);  }  ListNode\* insert\_first(ListNode\* head, int value) {  ListNode\* p = (ListNode\*)malloc(sizeof(ListNode));  p->data = value;  p->link = head;  return p;  }  ListNode\* insert(ListNode\* head, ListNode\* pre, element value) {  ListNode\* p = (ListNode\*)malloc(sizeof(ListNode));  p->data = value;  p->link = pre->link;  pre->link = p;  return head;  }  ListNode\* delete\_first(ListNode\* head) {  ListNode\* removed;  if (head == NULL) return NULL;  removed = head;  head = removed->link;  free(removed);  return head;  }  ListNode\* delete(ListNode\* head, ListNode\* pre) {  ListNode\* removed;  removed = pre->link;  pre->link = removed->link;  free(removed);  return head;  }  void print\_list(ListNode\* head) {  for (ListNode\* p = head; p != NULL; p = p->link)  printf("%d->", p->data);  printf("NULL \n");  }  int get\_length(ListNode\* head) {  int size = 0;  for (ListNode\* p = head; p != NULL; p = p->link) size++;  return size;  }  int main(void) {  ListNode\* head = NULL;  for (int i = 0; i < 5; i++) {  head = insert\_first(head, i);  print\_list(head);  }  printf("List의 Size = %d\n", get\_length(head));  return 0;  }  실행결과 |