|  |  |  |  |
| --- | --- | --- | --- |
| 2022/2 『자료구조』실습 보고서 | | | |
| 제목 | 7장 실습( O ) 과제( ) | 제출일자 | 2022.  11 .    09 . |
| 학번 | 201911608 | 이름 | 김지환 |

|  |
| --- |
| 1. 프로그램 7.9  #include <stdio.h>  #include <malloc.h>  typedef int element;  typedef struct StackNode {  element data;  struct StackNode\* link;  } StackNode;  typedef struct {  StackNode\* top;  } LinkedStackType;  void init(LinkedStackType\* s)  {  s->top = NULL;  }  int is\_empty(LinkedStackType\* s)  {  return (s->top == NULL);  }  int is\_full(LinkedStackType\* s)  {  return 0;  }  void push(LinkedStackType\* s, element item)  {  StackNode\* temp = (StackNode\*)malloc(sizeof(StackNode));  temp->data = item;  temp->link = s->top;  s->top = temp;  }  void print\_stack(LinkedStackType\* s)  {  for (StackNode\* p = s->top; p != NULL; p = p->link)  printf("%d->", p->data);  printf("NULL \n");  }  element pop(LinkedStackType\* s)  {  if (is\_empty(s)) {  fprintf(stderr, "스택이 비어있음\n");  exit(1);  }  else {  StackNode\* temp = s->top;  int data = temp->data;  s->top = s->top->link;  free(temp);  return data;  }  }  element peek(LinkedStackType\* s)  {  if (is\_empty(s)) {  fprintf(stderr, "스택이 비어있음\n");  exit(1);  }  else {  return s->top->data;  }  }  int main(void)  {  LinkedStackType s;  init(&s);  push(&s, 1); print\_stack(&s);  push(&s, 2); print\_stack(&s);  push(&s, 3); print\_stack(&s);  pop(&s); print\_stack(&s);  pop(&s); print\_stack(&s);  pop(&s); print\_stack(&s);  return 0;  }  실행결과    -> 스택구조 LIFO이다. |
| 2. 프로그램 7.12  #include <stdio.h>  #include <stdlib.h>  typedef int element;  typedef struct QueueNode {  element data;  struct QueueNode\* link;  } QueueNode;  typedef struct {  QueueNode\* front, \* rear;  } LinkedQueueType;  void init(LinkedQueueType\* q)  {  q->front = q->rear = 0;  }  int is\_empty(LinkedQueueType\* q)  {  return (q->front == NULL);  }  int is\_full(LinkedQueueType\* q)  {  return 0;  }  void enqueue(LinkedQueueType\* q, element data)  {  QueueNode\* temp = (QueueNode\*)malloc(sizeof(QueueNode));  temp->data = data;  temp->link = NULL;  if (is\_empty(q)) {  q->front = temp;  q->rear = temp;  }  else {  q->rear->link = temp;  q->rear = temp;  }  }  element dequeue(LinkedQueueType\* q)  {  QueueNode\* temp = q->front;  element data;  if (is\_empty(q)) {  fprintf(stderr, "스택이 비어있음\n");  exit(1);  }  else {  data = temp->data;  q->front = q->front->link;  if (q->front == NULL)  q->rear = NULL;  free(temp);  return data;  }  }  void print\_queue(LinkedQueueType\* q)  {  QueueNode\* p;  for (p = q->front; p != NULL; p = p->link)  printf("%d->", p->data);  printf("NULL\n");  }  int main(void)  {  LinkedQueueType queue;  init(&queue);  enqueue(&queue, 1); print\_queue(&queue);  enqueue(&queue, 2); print\_queue(&queue);  enqueue(&queue, 3); print\_queue(&queue);  dequeue(&queue); print\_queue(&queue);  dequeue(&queue); print\_queue(&queue);  dequeue(&queue); print\_queue(&queue);  return 0;  }  실행결과    -> FIFO이다. |
| 3. 연습문제 03번  원형 연결 리스트에서 특정한 값을 탐색하는 함수 search()를 작성하고 테스트하라. search()는 다음과 같은 원형을 가진다.  // 원형 연결 리스트 L에서 data를 가지고 있는 노드를 찾아서 반환한다.  ListNode\* search(ListNode \*L, element data);  #include <stdio.h>  #include <stdlib.h>  typedef int element;  typedef struct ListNode { // 노드 타입  element data;  struct ListNode\* link;  } ListNode;  void print\_list(ListNode\* head) {  ListNode\* p;  if (head == NULL) return;  p = head->link;  do {  printf("%d->", p->data);  p = p->link;  } while (p != head);  printf("%d->\n\n", p->data);  }  ListNode\* insert\_first(ListNode\* head, element data) {  ListNode\* node = (ListNode\*)malloc(sizeof(ListNode));  node->data = data;  if (head == NULL) {  head = node;  node->link = head;  }  else {  node->link = head->link;  head->link = node;  }  return head;  }  ListNode\* insert\_last(ListNode\* head, element data)  {  ListNode\* node = (ListNode\*)malloc(sizeof(ListNode));  node->data = data;  if (head == NULL) {  head = node;  node->link = head;  }  else {  node->link = head->link;  head->link = node;  head = node;  }  return head;  }  ListNode\* search(ListNode\* L, element data) {  ListNode\* p = L;  for (p = p->link; p != L; p = p->link)  if (p->data == data) return p;  return NULL;  }  int main(void)  {  ListNode\* head = NULL;  head = insert\_last(head, 20);  head = insert\_last(head, 30);  head = insert\_last(head, 40);  head = insert\_first(head, 10);  print\_list(head);  if (search(head, 20) == NULL) printf("20찾기 : 없습니다.\n");  else printf("20찾기 : 있습니다.\n");    return 0;  }  실행결과 |
| 4. 연습문제 07번  이중 연결 리스트에서 특정한 값을 탐색하는 함수 search()를 작성하고 테스트하라. search()는 다음과 같은 원형을 가진다.  //이중 연결 리스트 L에서 data를 가지고 있는 노드를 찾아서 반환한다.  DListNode \*search(DListNode \*L, element data);  #include <stdio.h>  #include <stdlib.h>  typedef int element;  typedef struct DListNode {  element data;  struct DListNode\* llink;  struct DListNode\* rlink;  } DListNode;  void init(DListNode\* phead)  {  phead->llink = phead;  phead->rlink = phead;  }  void print\_dlist(DListNode\* phead)  {  DListNode\* p;  for (p = phead->rlink; p != phead; p = p->rlink) {  printf("<-| |%d| |-> ", p->data);  }  printf("\n");  }  void dinsert(DListNode\* before, element data)  {  DListNode\* newnode = (DListNode\*)malloc(sizeof(DListNode));  newnode->data = data;  newnode->llink = before;  newnode->rlink = before->rlink;  before->rlink->llink = newnode;  before->rlink = newnode;  }  void ddelete(DListNode\* head, DListNode\* removed) {  if (removed == head) return;  removed->llink->rlink = removed->rlink;  removed->rlink->llink = removed->llink;  free(removed);  }  DListNode\* search(DListNode\* L, element data) {  DListNode\* p;  if (L == NULL) return NULL;  for (p = L->rlink; p != L; p = p->rlink) {  if (p->data == data) return p;  }  return NULL;  }  int main(void)  {  DListNode\* head = (DListNode\*)malloc(sizeof(DListNode));  init(head);  printf("추가 단계\n");  for (int i = 0; i < 5; i++) {  dinsert(head, i);  print\_dlist(head);  }  if (search(head, 2) == NULL) printf("2 는 존재하지 않습니다.\n");  else printf("2 는 존재합니다.\n");  printf("\n삭제 단계\n");  for (int i = 0; i < 5; i++) {  print\_dlist(head);  ddelete(head, head->rlink);  }  free(head);  head = NULL;  if (search(head, 2) == NULL) printf("2 는 존재하지 않습니다.\n");  else printf("2 는 존재합니다.\n");  return 0;  } |
| 5. 6장 LAB 리스트를 역순으로 만드는 연산  #include <stdio.h>  #include <stdlib.h>  typedef int element;  typedef struct {  element data;  struct ListNode\* link;  }ListNode;  ListNode\* insert\_first(ListNode\* head, element v) {  ListNode\* p = (ListNode\*)malloc(sizeof(ListNode));  p->data = v;  p->link = head;  head = p;  return head;  }  void print\_list(ListNode\* head) {  for (ListNode\* p = head; p != NULL; p = p->link) printf("%d->", p->data);  printf("NULL \n");  }  ListNode\* reverse(ListNode\* head) {  ListNode\* p, \* q, \* r;  p = head;  q = NULL;  while (p != NULL) {  r = q;  q = p;  p = p->link;  q->link = r;  }  return q;  }  int main(void) {  ListNode\* head1 = NULL;  ListNode\* head2 = NULL;  head1 = insert\_first(head1, 10);  head1 = insert\_first(head1, 20);  head1 = insert\_first(head1, 30);  print\_list(head1);  head2 = reverse(head1);  print\_list(head2);  return 0;  }  실행결과    -> reverse 함수에서  1 -> 2 -> 3 -> NULL 일 경우  r = NULL, q=1, p=2, q->link =NULL // q = 1->NULL  r = 1, q=2, p=3, q->link = 2->1->NULL  r = 2, q=3, p=NULL, q->link = 3->2->1->NULL  p가 NULL이므로 종료 |