|  |  |  |  |
| --- | --- | --- | --- |
| 2022/2 『자료구조』실습 보고서 | | | |
| 제목 | 4장 실습( O ) 과제( ) | 제출일자 | 2022.  10 .    13 . |
| 학번 | 201911608 | 이름 | 김지환 |

|  |
| --- |
| 1. 프로그램 5.1  #include <stdio.h>  #include <stdlib.h>  #define MAX\_QUEUE\_SIZE 5  typedef int element;  typedef struct {  int front;  int rear;  element data[MAX\_QUEUE\_SIZE];  } QueueType;  void error(char\* message)  {  fprintf(stderr, "%s\n", message);  exit(1);  }  void init\_queue(QueueType\* q)  {  q->rear = -1;  q->front = -1;  }  void queue\_print(QueueType\* q)  {  for (int i = 0; i < MAX\_QUEUE\_SIZE; i++) {  if (i <= q->front || i > q->rear)  printf(" | ");  else  printf("%d | ", q->data[i]);  }  printf("\n");  }  int is\_full(QueueType\* q)  {  if (q->rear == MAX\_QUEUE\_SIZE - 1)  return 1;  else  return 0;  }  int is\_empty(QueueType\* q)  {  if (q->front == q->rear)  return 1;  else  return 0;  }  void enqueue(QueueType\* q, int item)  {  if (is\_full(q)) {  error("큐가 포화상태입니다.");  return;  }  q->data[++(q->rear)] = item;  }  int dequeue(QueueType\* q)  {  if (is\_empty(q)) {  error("큐가 공백상태입니다.");  return -1;  }  int item = q->data[++(q->front)];  return item;  }  int main(void)  {  int item = 0;  QueueType q;  init\_queue(&q);  enqueue(&q, 10); queue\_print(&q);  enqueue(&q, 20); queue\_print(&q);  enqueue(&q, 30); queue\_print(&q);  item = dequeue(&q); queue\_print(&q);  item = dequeue(&q); queue\_print(&q);  item = dequeue(&q); queue\_print(&q);  return 0;  }  실행결과    ->  선형적인 순서로 10, 20, 30이 추가 되고  10, 20 이 앞에서 사라짐을 볼 수 있지만 30은 중간에 머무는 문제점을 확인할 수 있다. |
| 2. 프로그램 5.2  #define \_CRT\_SECURE\_NO\_WARNINGS  #include <stdio.h>  #include <stdlib.h>  #define MAX\_QUEUE\_SIZE 5  typedef int element;  typedef struct {  element data[MAX\_QUEUE\_SIZE];  int front, rear;  } QueueType;  void error(char\* message) {  fprintf(stderr, "%s\n", message);  exit(1);  }  void init\_queue(QueueType\* q) {  q->front = q->rear = 0;  }  int is\_empty(QueueType\* q) {  return (q->front == q->rear);  }  int is\_full(QueueType\* q) {  return ((q->rear + 1) % MAX\_QUEUE\_SIZE == q->front);  }  void queue\_print(QueueType\* q) {  printf("QUEUE(front=%d rear=%d) = ", q->front, q->rear);  if (!is\_empty(q)) {  int i = q->front;  do {  i = (i + 1) % (MAX\_QUEUE\_SIZE);  printf("%d | ", q->data[i]);  if (i == q->rear)  break;  } while (i != q->front);  }  printf("\n");  }  void enqueue(QueueType\* q, element item) {  if (is\_full(q))  error("큐가 포화상태입니다");  q->rear = (q->rear + 1) % MAX\_QUEUE\_SIZE;  q->data[q->rear] = item;  }  element dequeue(QueueType\* q) {  if (is\_empty(q))  error("큐가 공백상태입니다");  q->front = (q->front + 1) % MAX\_QUEUE\_SIZE;  return q->data[q->front];  }  element peek(QueueType\* q) {  if (is\_empty(q))  error("큐가 공백상태입니다");  return q->data[(q->front + 1) % MAX\_QUEUE\_SIZE];  }  int main(void) {  QueueType queue;  int element;  init\_queue(&queue);  printf("--데이터 추가 단계--\n");  while (!is\_full(&queue)) {  printf("정수를 입력하시오: ");  scanf("%d", &element);  enqueue(&queue, element);  queue\_print(&queue);  }  printf("큐는 포화상태입니다.\n\n");  printf("--데이터 삭제 단계--\n");  while (!is\_empty(&queue)) {  element = dequeue(&queue);  printf("꺼내진 정수: %d \n", element);  queue\_print(&queue);  }  printf("큐는 공백상태입니다.\n");  return 0;  }  실행결과    ->  enqueue dequeue  q->rear = (q->rear + 1) % MAX\_QUEUE\_SIZE; q->front = (q->front + 1) % MAX\_QUEUE\_SIZE;  q->data[q->rear] = item; return q->data[q->front];  enqueue와 dequeue 의 내용이 queue의 rear 와 front 값을 size의 나머지로 설정하면서  rear이 max == 5 가 되면 size는 5이므로 rear = 0으로 재설정된다. |
| 3. 프로그램 5.2 Quiz  01. 원형 큐에서 front와 rear이 가리키는 것은 무엇인가?  front는 영어사전 의미로 앞쪽이라는 뜻, rear은 뒤쪽이라는 뜻이다.  소스 내에서 front를 dequeue에서 index로 사용하므로 front는 앞쪽에 있는 index이다.  queue는 FIFO 구조로 먼저 입력된 값이 출력된다.  즉, 맨 앞에 있는 값이 출력되므로 front는 출력할 index를 가르킨다.  -> dequeue는 데이터 삭제 단계이다.  rear도 마찬가지로 enqueue에서 index로 사용하니 뒤쪽에 있는 index이다.  즉, 맨 뒤에 있는 값은 queue 구조에서 마지막에 삽입된 값이다.  rear은 queue에 삽입할 값의 위치를 가르키는 index이다.  -> enqueue는 데이터 추가 단계이다.  02. 크기가 10인 원형큐에서 front와 rear이 모두 0으로 초기화 되었다고 가정하고 다음과 같은 연산 후에 front와 rear의 값을 말하라.  enqueue(a), enqueue(b), enqueue(c), dequeue(), enqueue(d)  -> 01 내용을 토대로  1) enqueue 과정을 통해 rear의 값이 (real+1)%원형큐사이즈 로 변화하고  2) dequeue 과정을 통해 front의 값이 (front+1)%원형큐사이즈 로 변화함을 알 수 있다.  3) 또, init\_queue(&queue); 를 통해 rear과 front의 초기 값이 0임을 알 수 있다.  enqueue(a), enqueue(b), enqueue(c)  -> loop(3) : rear=((rear+1)%10)  -> loop(1) -- rear = (0+1)%10 = 1  -> loop(1) -- rear = (1+1)%10 = 2  -> loop(1) -- rear = (2+1)%10 = 3  dequeue()  -> front=((front+1)%10) = (0+1)%10 = 1  enqueue(d)  -> rear=((real+1)%10) = (3+1)%10 = 4  ---> 연산 후 front와 rear의 값 ..... front : 1, rear : 4 |
| 4. 프로그램 5.3  #include <stdio.h>  #include <stdlib.h>  #define MAX\_QUEUE\_SIZE 5  typedef int element;  typedef struct {  element data[MAX\_QUEUE\_SIZE];  int front, rear;  } QueueType;  void error(char\* message) {  fprintf(stderr, "%s\n", message);  exit(1);  }  void init\_queue(QueueType\* q) {  q->front = q->rear = 0;  }  int is\_empty(QueueType\* q) {  return (q->front == q->rear);  }  int is\_full(QueueType\* q) {  return ((q->rear + 1) % MAX\_QUEUE\_SIZE == q->front);  }  void queue\_print(QueueType\* q) {  printf("QUEUE(front=%d rear=%d) = ", q->front, q->rear);  if (!is\_empty(q)) {  int i = q->front;  do {  i = (i + 1) % (MAX\_QUEUE\_SIZE);  printf("%d | ", q->data[i]);  if (i == q->rear)  break;  } while (i != q->front);  }  printf("\n");  }  void enqueue(QueueType\* q, element item) {  if (is\_full(q))  error("큐가 포화상태입니다");  q->rear = (q->rear + 1) % MAX\_QUEUE\_SIZE;  q->data[q->rear] = item;  }  element dequeue(QueueType\* q) {  if (is\_empty(q))  error("큐가 공백상태입니다");  q->front = (q->front + 1) % MAX\_QUEUE\_SIZE;  return q->data[q->front];  }  element peek(QueueType\* q) {  if (is\_empty(q))  error("큐가 공백상태입니다");  return q->data[(q->front + 1) % MAX\_QUEUE\_SIZE];  }  int main(void) {  QueueType queue;  int element;  init\_queue(&queue);  srand(time(NULL));  int tmp;  for (int i = 0;i < 100; i++) {  tmp = rand();  if (tmp % 5 == 0)  enqueue(&queue, rand() % 100);  printf("rand = %d --> ", tmp);  queue\_print(&queue);  tmp = rand();  if (tmp % 10 == 0) {  int data = dequeue(&queue);  }  printf("rand = %d --> ", tmp);  queue\_print(&queue);  }  return 0;  }  실행결과    -> code에서 enqueue 나 dequeue 과정이 아닐 때도 출력해서 수정했습니다.  random 값이 5의 배수 일 때 enqueue, 10의 배수 일 때 dequeue 하는 모습을 직접 볼 수 있게 추가로 수정했습니다. |
| 5. 연습문제 09  09. 2개의 스택을 사용하여 큐를 구현할 수 있을까? 2개의 스택을 사용하여 큐를 구현해보자. 입력이 들어오면 스택 #1에 넣는다. 출력 요청이 들어오면 스택 #2에서 요소를 꺼낸다. 스택 #2가 비어 있을 때는 스택 #1의 모든 요소를 꺼내서 스택 #2에 넣는다. 프로그램으로 작성해보자.  #define \_CRT\_SECURE\_NO\_WARNINGS  #include <stdio.h>  #include <stdbool.h>  #define MAX 5  typedef struct {  int data[MAX];  int top;  }Stack;  void init(Stack\* s) {  s->top = -1;  }  bool full(Stack\* s) {  if (s->top == MAX-1) return true;  return false;  }  void push(Stack\* s, int item) {  s->data[++s->top] = item;  }  bool empty(Stack\* s) {  if (s->top == -1) return true;  return false;  }  int pop(Stack\* s) {  return s->data[s->top--];  }  int main() {  Stack s1, s2;  init(&s1), init(&s2);  int n;  while (!full(&s2)) {  printf("queue에 값 삽입 : ");  scanf("%d", &n);  push(&s1, n);  push(&s2, pop(&s1));  }  printf("queue 포화!\n");  printf("queue 출력!\n");  while (!empty(&s2)) {  printf("[%d]", pop(&s2));  }  return 0;  }  실행결과 |