자료구조와 알고리즘 10주차 보고서

20193281 송형준

1. 단순 연결 리스트

```
#include <iostream>

using namespace std;

class SNode{
	int data;
	SNode* link;
	friend class SLinkedList; //freind 문법 : friend class 클래스이름
};

class SLinkedList{
	SNode* head;
	SNode* tail;
	int count;
public:
	SLinkedList();
	~SLinkedList();
//생성
	SNode* makeSNode(const int& e);
//조사
	bool isEmpty() const;
	int countSNode();
	int searchSNode(const int& e);
//출력
	void PrintSLinkedList() const;
//삽입
	void insertRear(const int& e);
//삭제
	void removeFront();
	void removeMid(SNode* preSNode);
	void removeRear();
};

inline void error(const char* message){
	cout << message;
	exit(100); //exit :
}
```

```
//생성자 & 소멸자
SLinkedList::SLinkedList(){
	head = NULL;
	tail = NULL;
	count = 0;
}
SLinkedList::~SLinkedList(){
	while(!isEmpty()) removeRear();
}

//생성
SNode* SLinkedList::makeSNode(const int& e){
	SNode* s = new SNode;
	s->data = e;
	s->link = NULL;

	return s;
}

//조사
bool SLinkedList::isEmpty() const{
	return head==NULL;
}
int SLinkedList::countSNode(){
	return count;
}

int SLinkedList::searchSNode(const int& e){
	SNode* temp= head;
	count = 0;//count 0으로 초기화해두고 시작

	while(temp!=NULL){ //마지막 노드 while문 실행
		count++;
		if(temp->data == e) return count;
		else temp = temp->link;
	}//if-else 아니여도 똑같음
	//while의 결과 temp=null

	if(temp==NULL) return 0;
}
```

```
//출력
void SLinkedList::PrintSLinkedList() const{
	SNode* temp = head;

	while(temp!=NULL){
		cout << temp->data << " --> ";
		temp = temp->link;
	}

	if(temp==NULL) cout << "NULL";
}
```

```
//삽입

//앞부터 삽입
void SLinkedList::insertFront(const int& e){
    SNode* newSNode = makeSNode(e); //새로운 노드 생성

    //if(isEmpty()) head=newSNode else{}; //없어도 됨

    newSNode->link = head;//head값 물려받음
    head = newSNode;

}
//중간 삽입
void SLinkedList::insertMid(const int& e, SNode* preSNode){

    SNode* newSNode = makeSNode(e);//새로운 노드 생성

    if(isEmpty()){
        head = newSNode;
        tail = newSNode;
    }
    else{
        newSNode-> link = preSNode->link;
        preSNode->link = newSNode;
    }
}
//뒤부터 삽입
void SLinkedList::insertRear(const int& e){
    SNode* newSNode = makeSNode(e);

    if(isEmpty()){
        head = newSNode;
        tail = newSNode;
    }
    else{
        tail->link = newSNode;
        tail = newSNode;
    }

    count ++;
}
```

```
//삭제

//앞에서 삭제
void SLinkedList::removeFront(){
    if(isEmpty()) return;

    SNode* old = head;
    head = old->link;
    delete old;

    count--;
}
//중간에서 삭제
void SLinkedList::removeMid(SNode* preSNode){
    if(isEmpty()) return;

    SNode* old = preSNode->link;
    preSNode->link = old->link;
    delete old;

    count--;
}
//뒤에서 삭제
void SLinkedList::removeRear(){
    SNode* temp = head;

    if(isEmpty()) {
        cout << "빈 리스트입니다 " << endl;
        return;
    }

    if(temp == tail){//노드 하나일 때
        delete head;

        head = NULL;
        tail = NULL;
        count--;
        return;
    }

    SNode* old = tail;
    count = 1;
```

```
    //이전 노드 탐색
    while(temp->link!=tail){
        temp = temp->link;
        count++;
    }//while문 결과 : n-1 노드

    temp->link = NULL;
    tail = temp;
    delete old;
    count--;
}
//메인함수

int main(){
    int num;
    SLinkedList s = SLinkedList();
    while(true){
        cout << " 임의의 정수 입력(종료 : 0): ";
        cin >> num;
        if(num == 0 ) break;
        //맨 마지막 노드로 삽입
        s.insertRear(num);
    }

    // 전체 원소 출력
    if(s.isEmpty()){
        cout << "입력된 데이터가 없습니다..." << endl;
        return 0;
    }

    printf("₩n ### 입력된 데이터 ### ₩n₩n ");
    s.PrintSLinkedList();

    printf("₩n ### 데이터 갯수 ### ₩n₩n ");
    cout << "총 데이터 갯수 : " << s.countSNode() << endl;

    s.~SLinkedList();

    return 0;
}
```

연구조사

함수 뒤에 const 의 의미 : 읽기 전용 함수이다.

리스트의 구성 함수는 생성, 조사, 출력, 삽입, 삭제로 나누어 이해할 수 있다
생성 : makeDNode
조사 : isEmpty, countDNode, searchSNode
출력 : PrintSLinkedList
삽입 : insertRear
삭제 : deleteFront, deleteMid, deleteRear

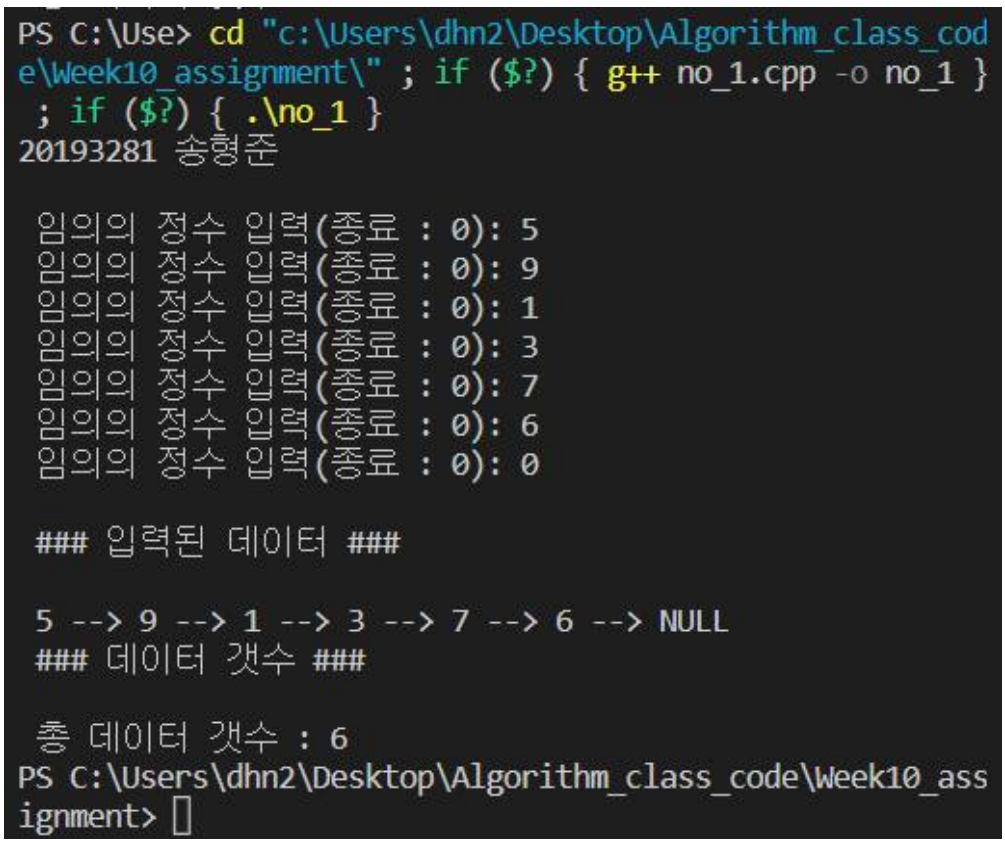
//삽입 알고리즘 과정 요약

1. 새로운 노드를 생성한다
2. head나 tail, pre_node 에서 물려받을 값 먼저 처리
3. 이후 head,tail,pre_node 갱신

//삭제 알고리즘 과정 요약
1. 살릴 정보 작업
2. 노드 삭제

** 뒤에서 삭제하는 경우, 노드가 한 개일 때에 대한 예외처리가 필요하다. 그 이유는 n-1번째의 노드에 대한 정보를 가져와야하는데, 노드가 한 개인 경우엔 n-1번째 노드가 존재하지 않기 때문이다.

코드 결과



2. 이중 연결 리스트

```
#include <iostream>
using namespace std;

inline void error(const char* message){
	cout << message << endl;
	exit(100);
}
using namespace std;

class DNode{
	int data;
	DNode* Llink;
	DNode* Rlink;
	friend class DLinkedList;
};

class DLinkedList{
	DNode* head;
	DNode* tail;
	int count;
public :
//생성자, 소멸자
	DLinkedList();
	~DLinkedList();
//생성
	DNode* makeDNode(const int& e);
//조사
	bool isEmpty() const;
	int countDNode();
//삽입
	void insertRear(const int& e);
//삭제
	void deleteFront();
	void deleteMid(DNode* pre_DNode);
	void deleteRear();
//출력
	void printBy_Rlink() const;
	void printBy_Llink() const;
};
```

```
//생성자 & 소멸자
DLinkedList::DLinkedList(){
	head = NULL;
	tail = NULL;
	count = 0;
}
DLinkedList::~DLinkedList(){
	while(!isEmpty()) deleteRear();
	//while(!isEmpty()) deleteFront();
	if(isEmpty()) cout << "NULL";

	exit(0);
}

//생성
DNode* DLinkedList::makeDNode(const int& e){
	DNode* nNode = new DNode;

	nNode->data=e;
	nNode->Llink=NULL;
	nNode->Rlink=NULL;
}
//조사
bool DLinkedList::isEmpty() const{
	return head == NULL;
}
int DLinkedList::countDNode(){
	return count;
}
//삽입
void DLinkedList::insertRear(const int& e){
	DNode* nNode = makeDNode(e); //노드 생성

	if(isEmpty()){ //빈 노드 확인
		head = nNode;
		tail = nNode;
	}
	else{
		tail->Rlink = nNode;
		nNode->Llink = tail;
		tail = nNode;
	}
	count ++;
```

```
}

//삭제
void DLinkedList::deleteFront(){

    if(isEmpty()) error("빈 리스트입니다.");

    DNode* old = head;
    old->Llink = NULL;
    head = old->Rlink;

    delete old;

    count--;
}

void DLinkedList::deleteMid(DNode* pre_DNode){

    if(isEmpty()) error("빈 리스트입니다.");

    DNode* old = pre_DNode->Rlink;//삭제할 노드


    pre_DNode->Rlink = old->Rlink;
    old->Rlink->Llink = pre_DNode;

    delete old;

    count--;
}

void DLinkedList::deleteRear(){

    if(isEmpty()) error("빈 리스트입니다.");

    DNode* old = tail; //삭제할 노드

    //노드 하나 남은 경우
    if(head == tail){
        head = NULL;
        tail = NULL;
    }
    else{
```

```
    // n-1 노드 Rlink -> NULL로
    // tail -> n-1로
        old->Llink->Rlink = NULL;
        tail = old->Llink;
    }

    delete old;

    count--;
}

//출력
void DLinkedList::printBy_Rlink() const{
    DNode* temp = head;

    while(temp){
        cout << temp->data << " ->> ";
        temp = temp->Rlink;
    }//마지막 노드도 while문 실행

    cout << "NULL" << endl;
}

void DLinkedList::printBy_Llink() const{
    DNode* temp = tail;

    while(temp){
        cout << temp->data << " ->> ";
        temp = temp -> Llink;
    }

    cout << "NULL" << endl;
}
```

```
//메인함수
int main(){
	int num;
	DLinkedList d = DLinkedList();

	while(true){
		cout << "임의의 정수 입력(종료 : 0) : ";
		cin >> num;

		if(num==0) break;

		d.insertRear(num);
	}

	if(d.isEmpty()){
		cout << "입력된 데이터가 없습니다..." << endl;
		return 0;
	}

	printf("%c",'₩n');
	cout << "### 입력된 데이터 ### " << endl;
	printf("%c",'₩n');

	d.printBy_Rlink();
	d.printBy_Llink();

	d.~DLinkedList();
}
```

연구조사

exit()
프로그램을 종료시키는 함수이다. exit(0)~exit(255)까지 가능하며,
exit(0)일 경우 프로그램이 정상적으로 종료되었음을 의미, exit(1)~exit(255)는 에러 발생으로 인해 프로그램이 종료되었음을 의미한다.

리스트의 구성 함수는 생성, 조사, 출력, 삽입, 삭제로 나누어 이해할 수 있다
생성 : makeDNode
조사 : isEmpty, countDNode, searchSNode
출력 : PrintSLinkedList
삽입 : insertRear
삭제 : deleteFront, deleteMid, deleteRear

//생성 알고리즘
1. 노드 동작할당 받기
2. data,Llink,Rlink 초기화

//삽입 알고리즘
1. 노드 생성 (n+1번째 노드)
2. n번째 노드의 Rlink = n+1번째 노드
3. n+1번째 노드의 Llink = n번째 노드
4. tail = n+1번째 노드

기존 tail는 n번째 노드에 대한 정보를 가지고 있으므로, n번째 노드 정보를 사용하는 모든 작업을 마친 뒤, tail을 n+1번째 노드로 바꾼다.

//삭제 알고리즘
deleteFront
head와 2번째 노드를 이어준 후, 1번째 노드를 삭제한다.

1. 빈 리스트 확인
2. old = 1번째 노드
3. 2번째 노드의 Llink = NULL
4. head = 2번째 노드
5. old 삭제

기존 head는 2번째 노드에 대한 정보를 알게 해준다. 따라서 반환하기 전에 2번째 노드에 대한 정보를 모두 사용한 후 반환한다.

deleteMid
k-1번째 노드와 k+1번째 노드를 이어준 후 , k번째 노드를 삭제한다.

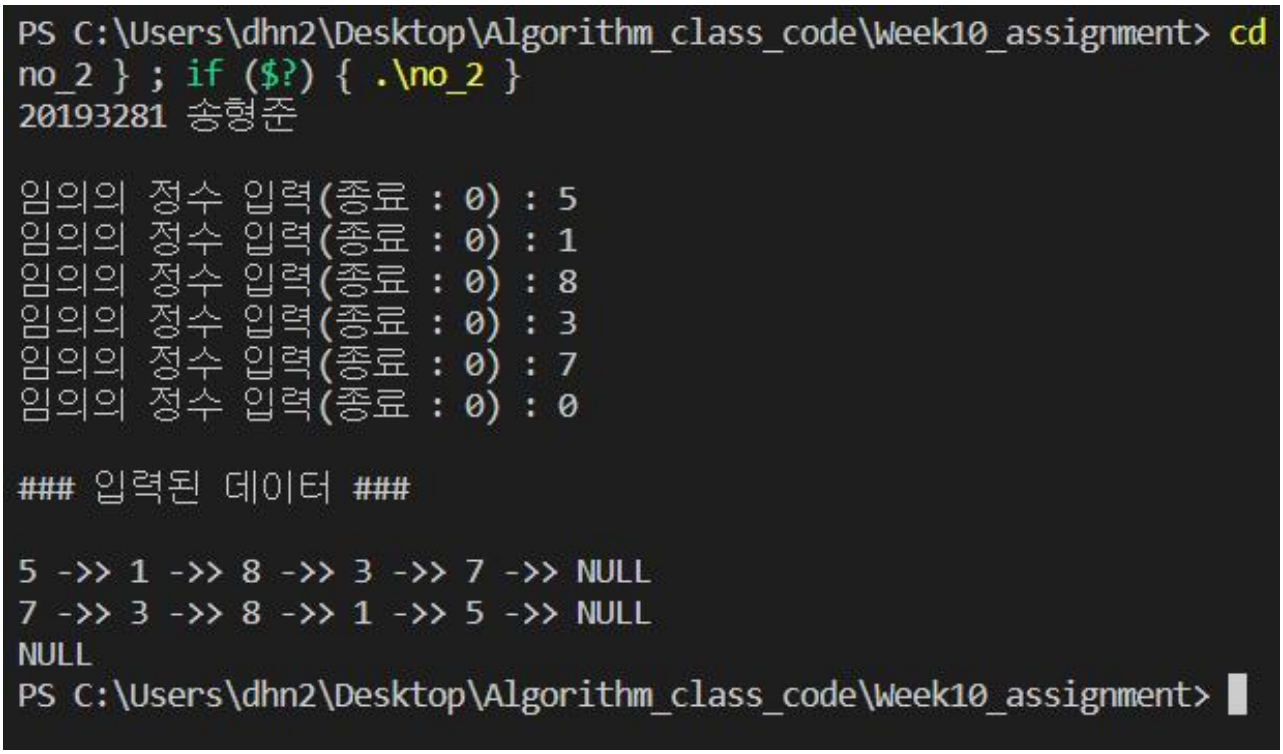
1. 빈 리스트 확인
2. old = k번째 노드

3. k-1번째 노드의 Rlink = k+1번째 노드
4. k+1번째 노드의 Llink = k-1번째 노드
5. old 삭제

deleteRear
n-1번째 노드와 tail을 이어준 후 , n번째 노드를 삭제한다.

1. 빈 리스트 확인
2. old = n번째 노드
3. 노드가 한 개일 경우 예외처리
3. n-1번째 노드의 Rlink = NULL
4. tail = n-1번째 노드
5. old 삭제

코드결과



3. 후위 표기법을 이용한 수식 계산

```
#include <iostream>
using namespace std;

#define     bufferMAXSIZE       1024

int         evalPostfix(char* str);
void        InfixToPostfix(char* postfix, char* infixStr);
int         isOperator(int op);
int         precedence(int op);

inline void error(const char* message);

template<typename E>
class stackNode{
    E data;
    stackNode<E>* link;
    template<typename V> friend class linkedStack;
};

template<typename E>
class linkedStack{
    stackNode<E>* top;
public:
//생성자 & 소멸자
    linkedStack();
    ~linkedStack();
//생성
    stackNode<E>* makeStackNode(const E& e);
//조사
    bool isEmpty();
//출력
    void printStack();
//push,pop,peek
    void push(const E& e);
    E pop();
    E peek();
};
```

```
template<typename E>
linkedStack<E>::linkedStack() : top(NULL){}

template<typename E>
linkedStack<E>::~linkedStack(){
    while(!isEmpty()) pop();
}
//생성
template<typename E>
stackNode<E>* linkedStack<E>::makeStackNode(const E& e){
    stackNode<E>* nNode = new stackNode<E>;

    nNode->data = e;
    nNode->link = NULL;

    return nNode;
}
//조사
template<typename E>
bool linkedStack<E>::isEmpty(){
    return top == NULL;
}
//출력
template<typename E>
void linkedStack<E>::printStack(){
    stackNode<E>* temp = top;

    cout << "₩n STACK [";
    while(temp){
        cout.width(3);
        cout << temp->data;
        temp = temp -> link;
    }
    cout << " ]" << endl;
}
```

```
//push,pop,peek
template<typename E>
void linkedStack<E>::push(const E& e){
    stackNode<E>* SNode = makeStackNode(e);

    SNode->link = top;
    top = SNode;
}
template<typename E>
E linkedStack<E>::pop(){
    if(isEmpty()) error("스택 공백 에러");

    stackNode<E>* old = top;
    E data = old ->data; //data 백업
    top = old->link;
    delete old;

    return data;
}

template<typename E>
E linkedStack<E>::peek(){
    if(isEmpty()) error("스택 공백 에러");

    return top->data;
}

inline void error(const char* message){
    cout << message << endl;
    exit(100);
}

//후위 표기법 함수
int evalPostfix(char* str){
    int op1, op2, res;
    char temp[bufferMAXSIZE], *p; //임시배열 : 두자리 수 이상도 처리하기 위해
    linkedStack<int> s;

    while(*str){
        //case 1 : 피연산자
        if(*str >= '0' && *str <= '9'){
            p = temp;
```

```
            while(*str >= '0' && *str <= '9')
                *p++ = *str++;
            *p++ = '₩0';
            s.push(atoi(temp));
        }
        //case 2: 연산자
        else if(isOperator(*str)){
            op2 = s.pop();
            op1 = s.pop();
            switch(*str){
                case '+' : s.push(op1 + op2); break;
                case '-' : s.push(op1 - op2); break;
                case '*' : s.push(op1 * op2); break;
                case '/' : s.push(op1 / op2); break;
            }
            str++;
        }
        else if(*str == ' ') str++;
        else{
            cout << "잘못된 수식입니다. " << endl;
            return 0;
        }
    }

    if(!s.isEmpty()) res = s.pop();

    return res;
}

void InfixToPostfix(char* postfix, char* infix){
    linkedStack<int> s;

    while(*infix){
        // case 1 : 괄호 '(' : 스택에 push
        if(*infix == '(') s.push(*infix++);
        // case 2 : 괄호 ')' : '('가 나올 때까지 pop 한 후에 ')'는 버린다.
        else if(*infix == ')'){
            while(s.peek() != '('){
                *postfix++ = s.pop();
                *postfix++ = ' ';
            }
            s.pop(); // '(' 를 버린다.
            infix++;
```

```
        }
        // case 3 : 연산자인 경우, 자신보다 우선순위가 높은 연산자는 스택에서 pop한 후, 자신을 push
한다.
        else if(isOperator(*infix)){
            while(!s.isEmpty() &&
            precedence(s.peek()) >= precedence(*infix))
            {
                *postfix++ = s.pop();
                *postfix = ' ';
            }
            s.push(*infix++);
        }
        // case 4 : 피연산자
        else if(*infix >= '0' && *infix <= '9'){
            while(*infix >= '0' && *infix <= '9')
                *postfix++ = *infix++;
            *postfix++ = ' ';
        }
        else if(*infix == ' ') infix++;
        else{
            cout << "잘못된 수식입니다. " << endl;
            return;
        }

    }

      while (!s.isEmpty()) {
            *postfix++ = s.pop();
            *postfix++ = ' ';
      }
      postfix--;
      *postfix = '₩0';

      return;
}
```

```
int   isOperator(int   op) {
            return op == '+' || op == '-' || op == '*' || op == '/';
}

int   precedence(int op) {
            if (op == '(') return 0;
            else if (op == '+' || op == '-') return 1;
            else if (op == '*' || op == '/') return 2;
            else return 3;
}

int main(){
            int    res;
            char infixStr[bufferMAXSIZE], postfixStr[bufferMAXSIZE];

            cout << "수식 입력: ";
            cin.getline(infixStr, bufferMAXSIZE);   // getline(cin, infixStr);

      cout << "입력된 수식 : " << infixStr << endl;

            InfixToPostfix(postfixStr, infixStr); //infixstr의 수식을 postfixstr로 변환
            cout << "₩n후위표기법 변환: " << postfixStr << endl;

            res = evalPostfix(postfixStr);

            cout << "연산결과: " << res << endl;

            return 0;
}
```

연구조사

push
새 노드를 추가하는 함수
1. 새 노드 생성
2. top이랑 연결

pop
노드를 삭제하는 함수
1. old에 삭제할 노드(n)를 백업
2. 노드를 삭제한 후에도, data를 return 할 수 있도록 백업
3. top을 n-1 노드로
4. 노드 삭제
5. 삭제한 노드의 data 리턴

peek
top에 있는 데이터 보여주는 함수

//후위연산자
먼저 중위표기법을 후위표기법으로 변환한 후, 이를 계산한다

evalPostfix
후위표기법 계산하는 함수

세가지 case
case 1: 피연산자
    (1) 숫자가 아닌 것을 만날 때까지 임시배열 temp에 숫자를 저장함
    (2) (1)이 끝나면 맨 뒤에 NULL문자 추가해주고, atoi를 통해 정수화시켜 push한다.
case 2 : 연산자
    (1) 스택에서 두 개의 정수를 pop
    (2) switch문 이용, 연산자에 따른 계산을 수행한 후, 스택에 push해둔다.
cass 3: 공백
    다음으로 바로 넘어간다

str++에 의해 문자를 하나씩 탐색, 마자막 문자인 NULL문자 만나면 반복문 종료 후 함수 종료

InfixToPOstfix
중위표기법을 후위표기법으로 변환해주는 함수

postfix : 후위표기법 배열
infix : 중위표기법 배열

네 가지의 case를 가진다

case 1: 괄호 '(' : 스택에 바로 push한다.
case 2: 괄호 ')' : '('가 나올 때까지 pop 한 후, ')'는 버린다.
case 3: 연산자 : 자신보다 우선순위 빠른 연산자는 pop 후 자기자신 push
case 4: 피연산자 : 바로 postfix 배열로 넣는다.

스택에는 괄호와 연산자만 들어가며, 피연산자는 바로 후위표기법 배열로 들어간다.

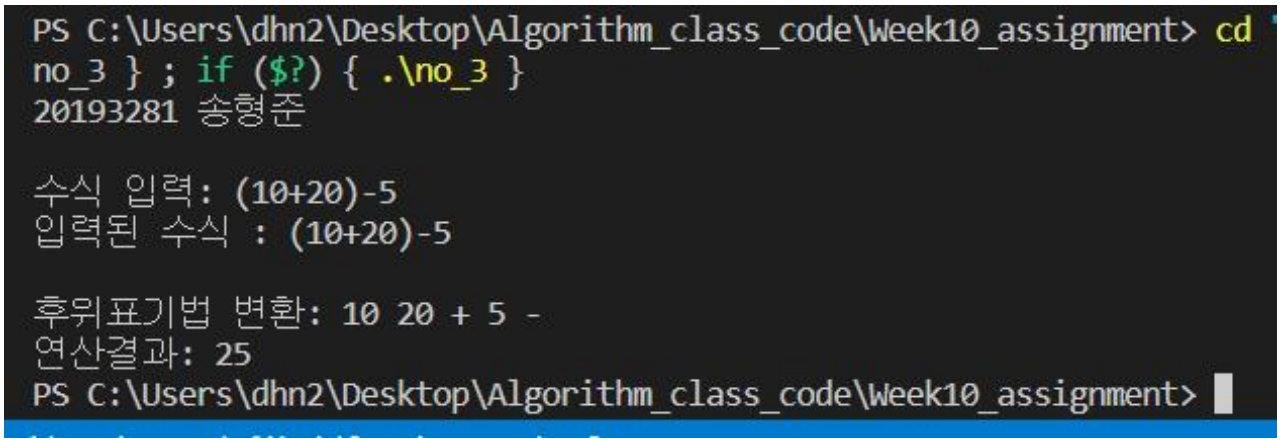
while(*infix) 종료 후엔, 스택에 남은 연산자나 괄호를 pop하여 postfix에 넣어준다.
필요한 이유 : ex) 3*(4+5) 의 경우, (4+5)는 처리되나, 스택에 *가 남는다.

isOperator
어떤 연산자인지 판별한다.

precedence
연산자 우선순위를 숫자로 표현한다.

'('의 우선순위는 반드시 필요하며, 최하위 순위여야 한다.
이유 :
'('의 우선순위를 정해두지 않으면, 스택 내에서 '('를 만났을 때, 이를 넘어갈 수 없다.
최하위 순위여야 하는 이유는, '('의 순위가 다른 연산자보다 순위가 높을 경우, pop되어버리기 때문이다.

코드 결과



4. 원형 큐 구현

```
#include <iostream>
using namespace std;

#define queueMAXSIZE 100

template <typename E>
class arrayQueue {
private:
	E queue[queueMAXSIZE];
	int front, rear, count;
public:
//생성자 & 소멸자
	arrayQueue();
	~arrayQueue();
//조사
	bool isEmpty(void) const;
	bool isFull(void) const;
//출력
	void printQueue(void) const;
//enQueue,deQueue,peek
	void enQueue(const E& e);
	E deQueue(void);
	E peek(void) const;
};

template<typename E>
arrayQueue<E>::arrayQueue(){
	front = -1;
	rear = -1;
	count = 0;
}

template<typename E>
arrayQueue<E>::~arrayQueue(){}

template<typename E>
bool arrayQueue<E>::isEmpty() const{
	return front == rear;
}
```

```
template<typename E>
bool arrayQueue<E>::isFull() const{
    return count == queueMAXSIZE;
    //return (rear+1)%queueMAXSIZE == front;
}

template<typename E>
void arrayQueue<E>::printQueue() const{

    if(isEmpty()) cout << "큐가 비어있습니다. " << endl;
    else{
        if(rear > front){
            for(int i=front+1; i<=rear;i++){
                cout.width(3);
                cout << queue[i];
            }
        }
        else if(rear < front){
            for(int i=front+1; i<queueMAXSIZE;i++){
                cout.width(3);
                cout << queue[i];
            }
            for(int i=0;i<=rear;i++){
                cout.width(3);
                cout << queue[i];
            }
        }

        cout << endl;
    }
}

template<typename E>
void arrayQueue<E>::enQueue(const E& e){
    if(isFull())
        cout << "큐가 꽉 찼습니다." << endl;
    else{
        rear = (rear+1)%queueMAXSIZE;
        queue[rear] = e;
        count ++;
        cout << "현재 count : " << count << endl;
    }
```

```
}
template<typename E>
E arrayQueue<E>::deQueue(){
    if(isEmpty()) cout << "큐가 비어있습니다. " << endl;
    else{
        count --;
        cout << "현재 count : " << count << endl;
        return queue[++front];
    }
}

template<typename E>
E arrayQueue<E>::peek() const{
    if(isEmpty()) cout << "큐가 비어있습니다. " << endl;
    else return queue[front+1];
}

int main(void)
{

    cout << "20193281 송형준₩n" << endl;

        int    num, choice;
    arrayQueue<int> aQ;

        while (true) {
            system("cls");
            cout << "₩n ### 큐 구현: 단순 연결 리스트 ### ₩n" << endl;
            cout << "1) 데이터 삽입: enQueue" << endl;
            cout << "2) 데이터 삭제: deQueue" << endl;
            cout << "3) 전체 출력" << endl;
            cout << "4) 프로그램 종료 ₩n" << endl;
            cout << "메뉴 선택 : ";
            cin >> choice;

            switch (choice) {
                case 1: cout << "₩n삽입 할 데이터 입력: ";
                  cin >> num;
                  aQ.enQueue(num);
                  break;
                case 2: cout << "삭제 된 데이터: " << aQ.deQueue() << endl;
                  break;
                case 3: aQ.printQueue();
```

```
                    break;
                case 4: cout << "프로그램 종료..." << endl;
                    return 0;
                default: cout << "잘못 선택 하셨습니다." << endl;
            }
            system("pause");
        }

        return 0;
}
```

연구조사

큐는 아래와 같이 이해할 수 있다.
생성자&소멸자
생성-makeQueueNode
탐색-isEmpty(), isFull() (X)
출력-printStack
큐함수- enqueue, dequeue, peek

isEmpty
큐가 비어있는지 확인한다. front와 rear가 같으면 큐가 비었다고 판단한다.

isFull
큐가 꽉 찼는지 확인한다. 선형 큐의 경우, rear+1의 값이 queueMaXSIZE와 같은지 확인하면 되나,
원형 큐는 같은 방법으로 할 경우, front = rear인 경우가 생겨, isEmpty와 구분할 수 없다.
따라서 별도의 count 변수를 유지하여 데이터의 갯수를 세고, count == queueMaXSIZE와 같은지 확인한다.

다른 방법으로는, 한칸의 큐를 쓰지 않는 방법이 있는데, 이 때는 (rear+1)%queueMaXSIZE가 front와 같은지 확인하는 방법을 사용한다

printQueue
큐의 모든 데이터를 확인한다.
rear > front인 경우는 선형 큐와 같은 상황이므로 front+1부터 rear까지 출력하면 되나,
rear < front인 경우는 queue의 마지막 칸까지 채워진 상태에서, 비어있는 앞칸에 데이터를 삽입하는 경우이므로 두 부분을 나누어 출력해준다.

enQueue
rear에 data를 입력한다.

(rear+1)%queueMAXSIZE의 의미 : rear값을 반드시 0~7 사이값으로 만든다.
이는 rear가 queueMAXSIZE-1 의 인덱스에 도달할 경우, rear를 다시 0부터 시작하도록 해주며, 앞칸부터 다시 삽입할 수 있도록 해준다

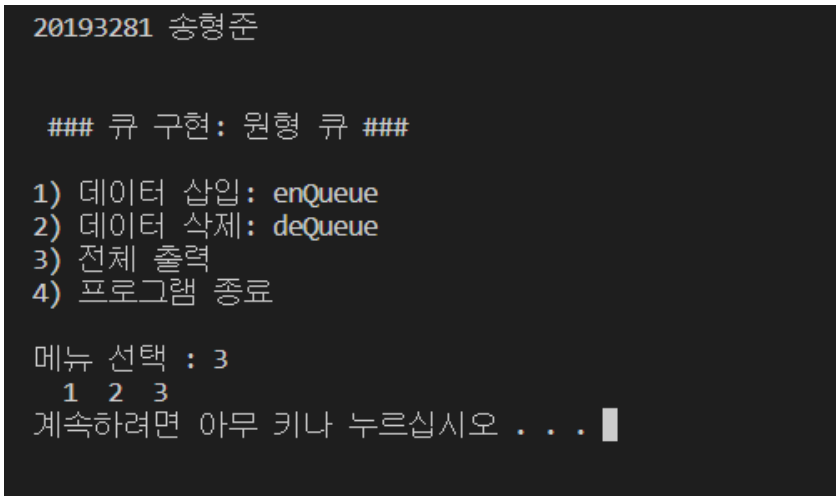
1. 꽉 차있는지 확인
2. 꽉 차지 않은 경우 rear = (rear+1)%queueMAXSIZE로 삽입할 자리 선정
3. 이후 삽입 실행
4. count 갯수 증가

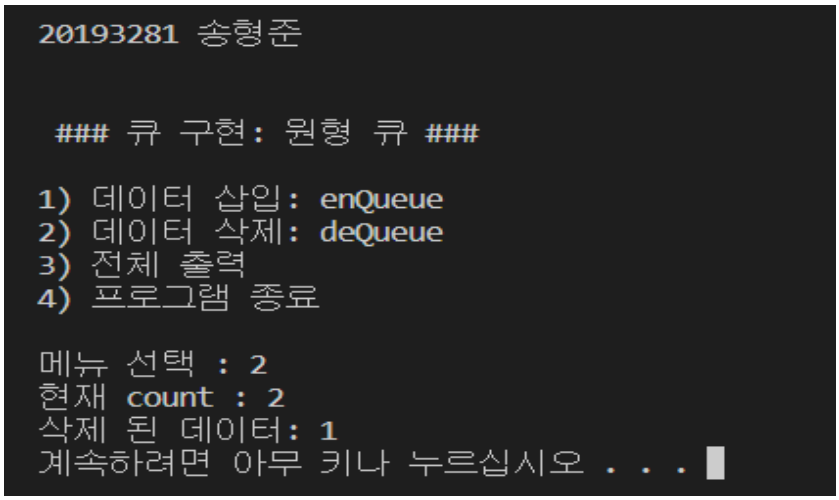
deQueue
front에서 data를 삭제한다.

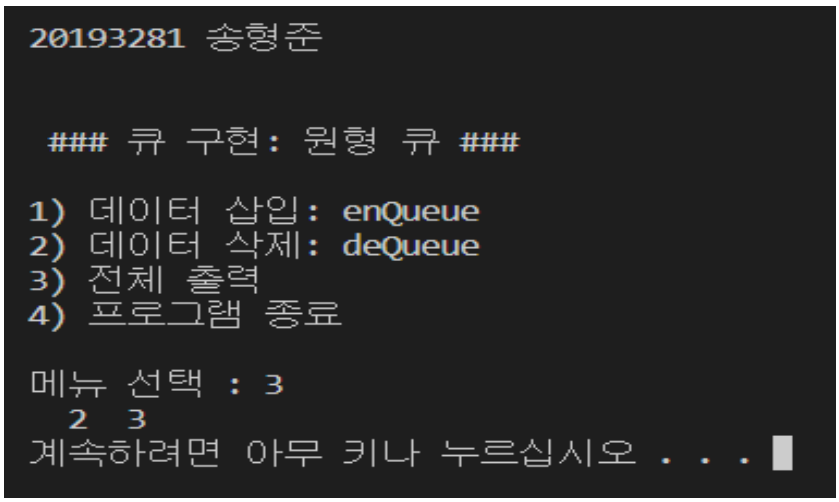
1. 비어있는지 확인
2. 비어있지 않은 경우, count 갯수 감소 후 , 현재 front의 다음칸을 반환

peek
front에 data를 반환한다.

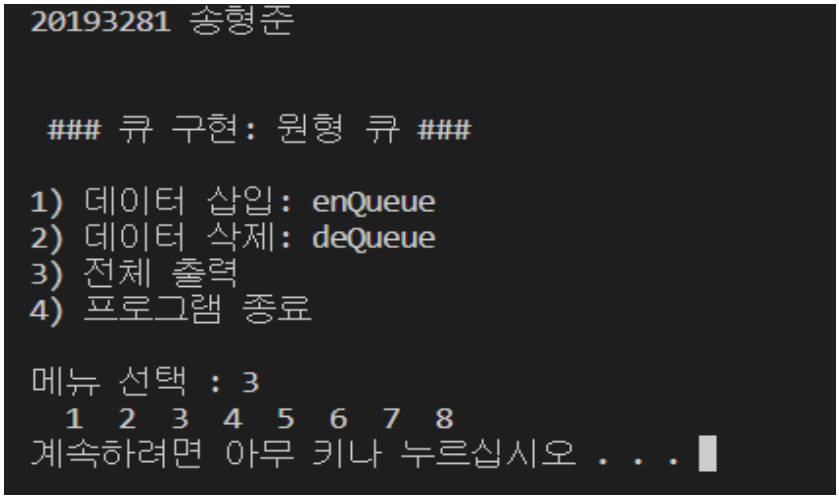
코드 결과

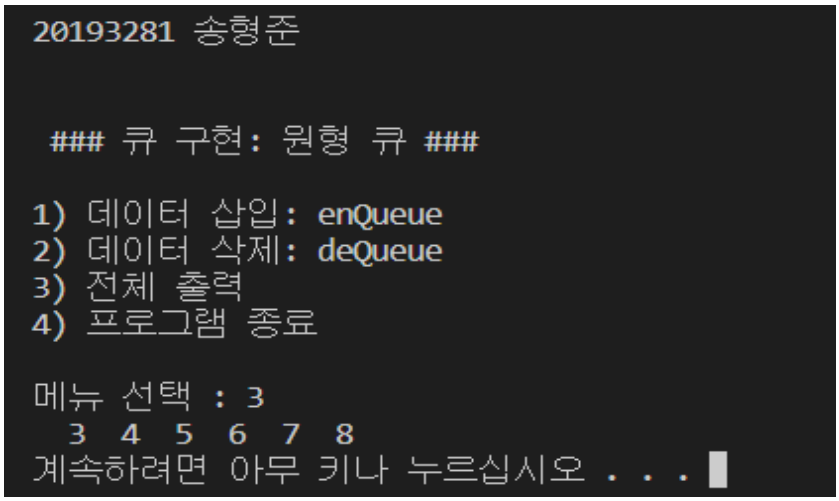


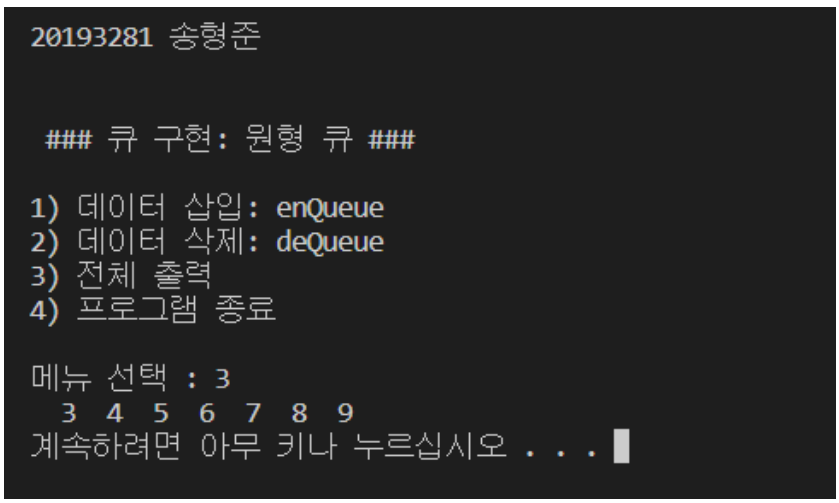




삭제 잘 되는 모습







배열 사이즈를 8로 한 후, 1~8로 배열을 채웠다.
이후 2개의 데이터를 삭제하고, 데이터 9를 삽입한 모습
정상적으로 작동하는 것을 확인할 수 있다.

5. 연결리스트 큐 구현

```
#include <iostream>
using namespace std;

inline void error(const char* message);

template<typename E>
class QueueNode{
	E data;
	QueueNode<E>* link;
	template<typename V> friend class LinkedQueue;
};

inline void error(const char* message){
	cout << message;
	exit(100); //exit :
}

template<typename E>
class LinkedQueue{
	QueueNode<E>* front,*rear; //E로 만들어진 front,rear;
public:
//생성자 & 소멸자
	LinkedQueue();
	~LinkedQueue();
//생성
	QueueNode<E>* makeQueueNode(const E& e);
//조사
	bool isEmpty() const;
//출력
	void printQueue() const;
//삽입,삭제,peek
	void enQueue(const E& e);
	E deQueue();
	E peek() const;
};


//생성자 & 소멸자
template<typename E>
LinkedQueue<E>::LinkedQueue(){
	front = NULL;
```

```
    rear = NULL;
}
template<typename E>
LinkedQueue<E>::~LinkedQueue(){}

//생성
template<typename E>
QueueNode<E>* LinkedQueue<E>::makeQueueNode(const E& e){
    QueueNode<E>* nNode = new QueueNode<E>;
    nNode->data = e;
    nNode->link = NULL;

    return nNode;
}

//조사
template<typename E>
bool LinkedQueue<E>::isEmpty() const{
    return front == NULL;
}

//출력
template<typename E>
void LinkedQueue<E>::printQueue() const{

    if(isEmpty()){
        cout << " Queue is Empty " << endl;
        return;
    }

    QueueNode<E>* temp = front; // 첫 노드부터

    while(temp){
        cout.width(3);
        cout << temp->data;
        temp = temp->link;
    }//while문의 결과 : NULL
    cout << '₩n';
}

template<typename E>
void LinkedQueue<E>::enQueue(const E& e){
    QueueNode<E>* nNode = makeQueueNode(e); //새로운 노드 생성
```

```
    if(isEmpty()){ //비어있는 리스트라면
        front = nNode;
        rear = nNode;
    }
    else{
        rear -> link = nNode; // n번째 노드가 new 노드를 가리키도록 함
        rear = nNode; // rear에 new 노드 저장
    }
}

template<typename E>
E LinkedQueue<E>::deQueue(){
    if(isEmpty()) error("큐 공백 에러");

    QueueNode<E>* old = front; //기존 front
    E data = old->data; // data 살리고
    front = old->link; // front 갱신
    free(old);

    return data;
}

template<typename E>
E LinkedQueue<E>::peek() const{
    if(isEmpty()){
        error("큐 공백 에러");
        return;
    }
    else return front->data;
}

int main(void)
{

    cout << "20193281 송형준₩n" << endl;

        int    num, choice;
        LinkedQueue<int>   s;

        while (true) {
            system("cls");
            cout << "₩n ### 큐 구현: 단순 연결 리스트 ### ₩n" << endl;
```

```
        cout << "1) 데이터 삽입: enQueue" << endl;
        cout << "2) 데이터 삭제: deQueue" << endl;
        cout << "3) 전체 출력" << endl;
        cout << "4) 프로그램 종료 ₩n" << endl;
        cout << "메뉴 선택 : ";
        cin >> choice;

        switch (choice) {
            case 1: cout << "₩n삽입 할 데이터 입력: ";
              cin >> num;
              s.enQueue(num);
              break;
            case 2: cout << "삭제 된 데이터: " << s.deQueue() << endl;
              break;
            case 3: s.printQueue();
              break;
            case 4: cout << "프로그램 종료..." << endl;
              return 0;
            default: cout << "잘못 선택 하셨습니다." << endl;
        }
        system("pause");
    }

    return 0;
}
```

연구조사

선입 선출

스택은 "후입선출"로 항상 top에서 push와 pop을 수행한다.
반면 큐는 "선입선출"으로 rear에서 push하고 front에서 pop을 수행한다.

아래와 같은 구조로 이해할 수 있다.
생성자&소멸자
생성-makeQueueNode
탐색-isEmpty(), //// isFull() (X)
출력-printStack
stack함수 - enqueue, dequeue, ,peek

enqueue
큐에 원소를 삽입한다.
rear->link : 기존의 맨 뒤가 새로운 노드를 가리키도록
rear = nNode : 마지막 노드 갱신

deQueue
큐에서 원소를 삭제한다

1. old에 삭제할 노드(front)를 백업
2. 노드를 삭제한 후에도, data를 return 할 수 있도록 백업
3. 2번째 노드를 front로
4. 노드 삭제
5. 삭제한 노드의 data 리턴

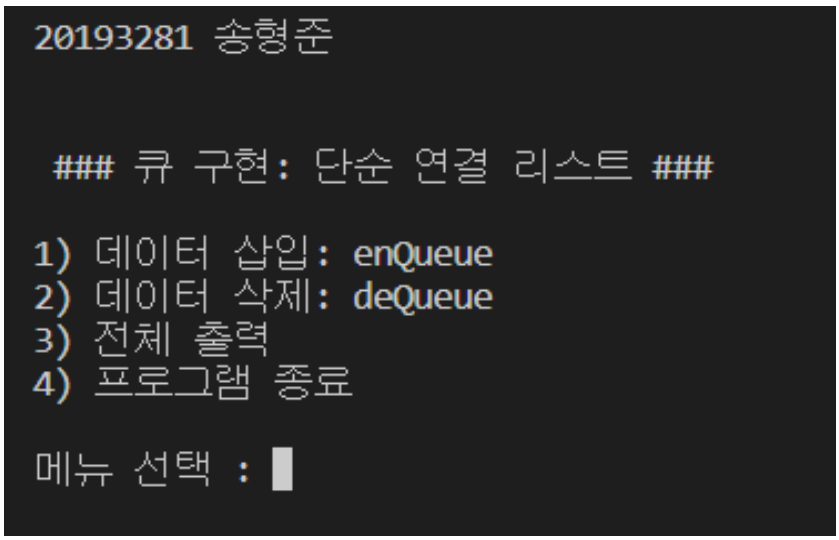
peek
큐에 가장 먼저 들어온 data를 확인

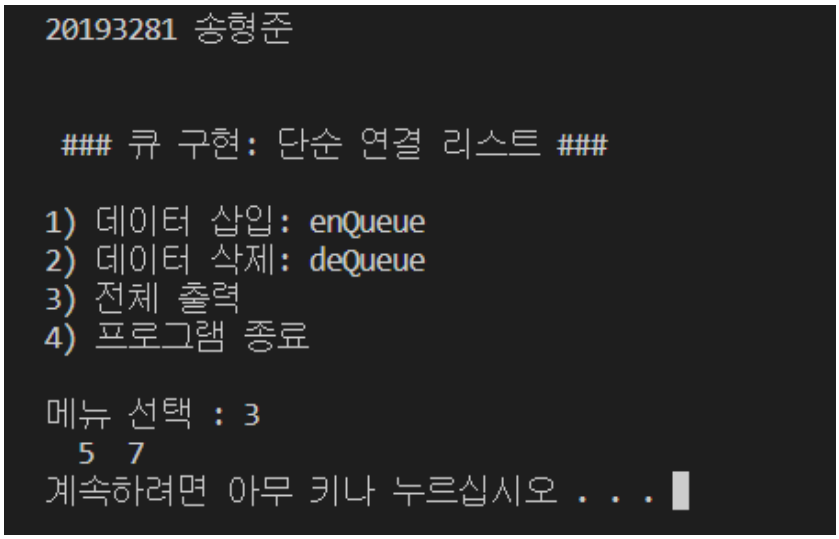
1. 빈 큐인지 확인
2. 비어있지 않으면 front의 데이터 반환

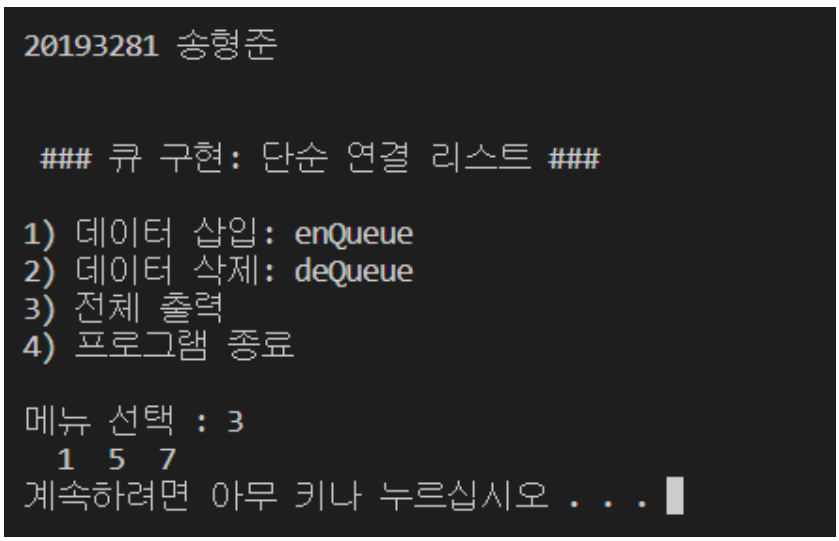
system("cls") : 실행됐던 기존 콘솔 화면을 삭제한다.
system("pause") : 콘솔 화면을 멈추는 함수. 원활하게 결과를 확인하고자 하는 부분에 넣어준다.

switch문의 입력값에는 정수형만 가능하다. 입력값으로 문자열을 넣을 경우, 이해할 수 없는 오류가 발생한다.

코드 결과


20193281 송형준
큐 구현: 단순 연결 리스트
1) 데이터 삽입: enQueue
2) 데이터 삭제: deQueue
3) 전체 출력
4) 프로그램 종료
메뉴 선택 :


20193281 송형준
큐 구현: 단순 연결 리스트
1) 데이터 삽입: enQueue
2) 데이터 삭제: deQueue
3) 전체 출력
4) 프로그램 종료
메뉴 선택 : 3
5 7
계속하려면 아무 키나 누르십시오 . . .


20193281 송형준
큐 구현: 단순 연결 리스트
1) 데이터 삽입: enQueue
2) 데이터 삭제: deQueue
3) 전체 출력
4) 프로그램 종료
메뉴 선택 : 3
1 5 7
계속하려면 아무 키나 누르십시오 . . .