스택을 이용한 알고리즘

1. 스택의 정의

   한쪽 끝(top)에서 삽입과 삭제가 일어나는 선형리스트

   LIFO(Last In First Out) 구조

   예)

2. 스택의 연산

   현재 스택의 상태를 나타내는 변수 top,

   데이터를 담을 배열 stack이 필요하다.

   (1) 삽입

   스택이 full한지 check

   top++, stack[top] = data

   (2) 삭제

   스택이 empty인지 check

   데이터 꺼내 사용, top--

수식 표기방식의 변경

수식 표기방식 (연산자의 위치에 따라)

중위표기(infix notation) 5 + 8 / 2

후위표기(postfix notation) 5 8 2 / +

왜 compiler는 중위표기식에서 후위표기식으로 변경해야 할까?

중위표기식에서는 연산자의 우선순위 때문에 왼쪽에서 오른쪽으로 연산이 진행되지 않는다.

중위표기식은 괄호를 가지고 있다.

변경방법

입력으로 주어진 중위표기식을 읽어가며 처리한다.

스택을 활용하여 연산자의 우선순위를 반영한다.

결과로서 후위표기식을 출력한다.

수식 표기방식의 변경 규칙(1)

변경 규칙(괄호가 없는 중위표기식)

다음의 규칙에 의해 입력으로 주어진 중위표기식을 읽어가며 후위표기식을 출력해 낸다.

(1) 피연산자를 만나면 그대로 출력한다.

(2) 스택이 비어 있을 때 만나는 연산자는 무조건 스택에 add한다.

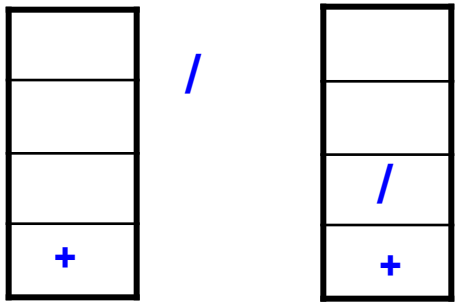
(3) 지금 처리하려는 연산자가 스택의 top의 연산자 보다 우선순위가 높으면 스택에 add한다.
아니면, 스택의 연산자를 delete 하여 출력한다.

(4) 표기식을 끝까지 다 읽으면 스택의 연산자를 모두 delete하여 출력한다.

예제1) 중위표기식 : 5 + 8 / 2

후위표기식 :

5 8 2 / +



수식 표기방식의 변경 규칙(2)

변경 규칙(괄호가 있는 중위표기식)

앞의 규칙을 적용하면서 괄호가 나오면 다음의 규칙을 적용한다.

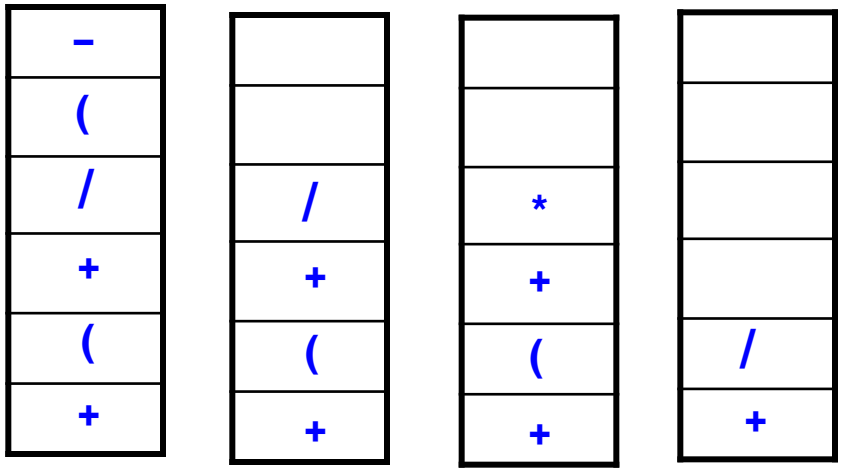
(1) 왼쪽 괄호는 스택에 들어올 때는 무조건 add 되도록 하고

일단 스택에 들어오면 우선순위가 가장 낮아져서 다음 들어오려는 연산자를 add한다.

(2) 오른쪽 괄호는 왼쪽 괄호가 나올 때까지 스택 안의 모든 연산자를 출력한다.

예제2)

중위표기식 : 2 + (4 + 8 / (3 – 1) * 5) / 3



2 4 8 3 1 - / 5 * + 3 / +

후위표기식의 계산

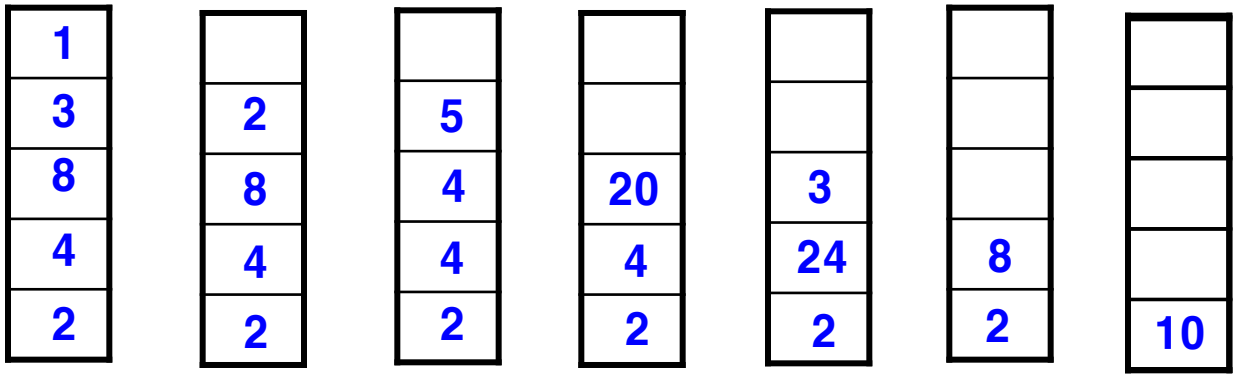
스택을 이용한 후위표기식의 계산과정 simulation

expr : 2 + (4 + 8 / (3 – 1) * 5) / 3

⇩ 중위표기식->후위표기식

2 4 8 3 1 - / 5 * + 3 / +

⇩ 계산



후위표기식의 계산 알고리즘

문자배열에 저장된 후위표기식을 읽어가면서

1) 피연산자는 스택에 넣는다(add_stack)

2) 연산자를 만나면 스택으로 부터 2개의 피연산자(op2, op1)를 꺼내(delete_stack)
그 결과를 스택에 넣는다. (add_stack)

위의 1) 2)의 과정을 주어진 후위표기식의 끝에 도달할 때까지 하면

스택에 마지막으로 남은 값이 수식을 계산한 값이된다.

후위표기식의 계산 함수

```
int cal(void)
{
    char symbol;
    int op1, op2, n=0;

    top = -1;
    symbol = pexpr[n++];
    while (symbol != '\0') {
        if (is_operator(symbol)) {
            op2 = delete_stack();
            op1 = delete_stack();
            switch (symbol) {
                case '+' : add_stack(op1+op2);
                        break;
                case '-' : add_stack(op1-op2);
                        break;
                case '*' : add_stack(op1*op2);
                        break;
                case '/' : add_stack(op1/op2);
            }
        }
        else
            add_stack(symbol-'0');   // 스택에 삽입
        symbol = pexpr[n++];
    }
    return delete_stack();  //결과 반환
```

후위표기식의 계산 함수 시뮬레이션

2 * ((3 + 9) / 3 + 4) / 2 → 2 3 9 + 3 / 4 + * 2 /

n	symbol	cal()에서 수행한 함수	top
0, 1, 2	'2', '3', '9'	add_stack(2), add_stack(3), add_stack(9)	2
3	'+'	2delete_stack() => 9, 3, add_stack(12)	1
4	'3'	add_stack(3)	2
5	'/'	2delete_stack() => 3, 12, add_stack(4)	1
6	'4'	add_stack(4)	2
7	'+'	2delete_stack() => 4, 4, add_stack(8)	1
8	'*'	2delete_stack() => 8, 2, add_stack(16)	0
9	'2'	add_stack(2)	1
10	'/'	2delete_stack() => 2, 16, add_stack(8)	0

노드기반으로 구현한 연결리스트

순차(인접)리스트 구조에서 연결 리스트 구조로

- 순차리스트는 자료구조 안의 데이터가 메모리에 연속적으로 저장되고
  그 순서에 의하여 데이터가 처리됨.
  배열의 인덱스 --- for문의 LCV
- 배열로 구현된 순차리스트는 실행 전에 그 크기가 정해져 있어야 함.
- 크기가 가변적인 자료구조인 경우?
- 데이터가 중간에 삽입되는 경우
- 자료구조의 중간으로부터 데이터가 삭제되는 경우

데이터의 크기와 처리가 동적인 환경에서는 필요할 때 마다
실행시간에 노드를 할당하여 연결해서 사용하는 구조가 적합하다.

연결리스트의 개념

◆ 연결 리스트의 구조

- [데이터, 링크]의 형태의 노드를 기본 단위로 연결되어 있음

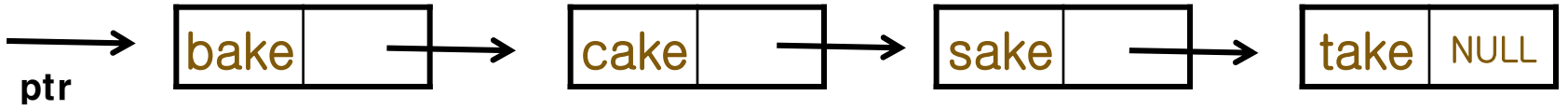
데이터	링크

- 데이터 필드(data field) : 표현하려는 값을 저장
- 링크 필드(link field) : 다음 노드의 주소를 저장

단순 연결 리스트 (singly linked list)

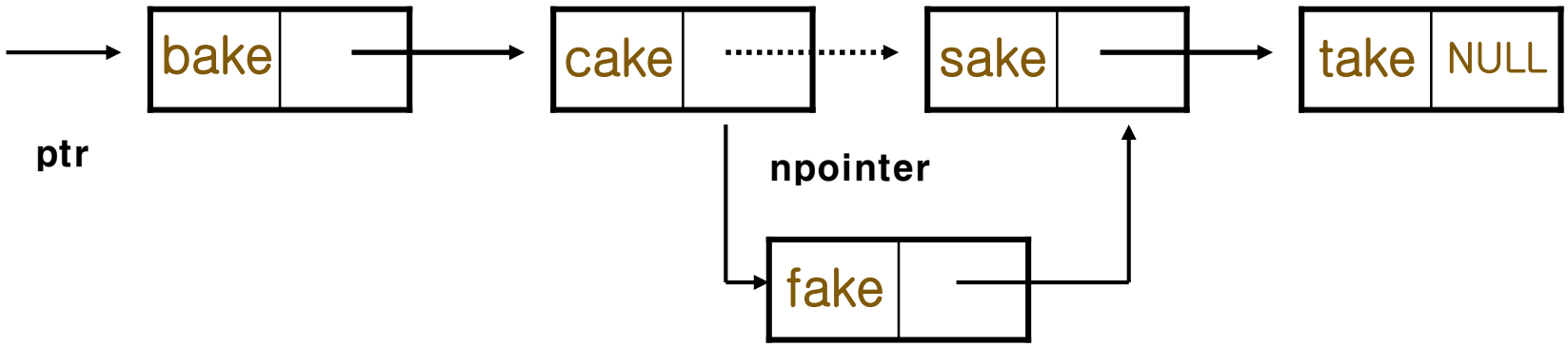
◆ 단순 연결리스트의 개념

- ptr이 가리키는 첫 노드로 부터 연속적으로 링크를 따라 데이터가 저장되고
- 마지막 노드의 링크는 NULL이 된다.
- 예) {bake” , “cake” , “sake” , “take” } 을 알파벳 순서로 저장할 때



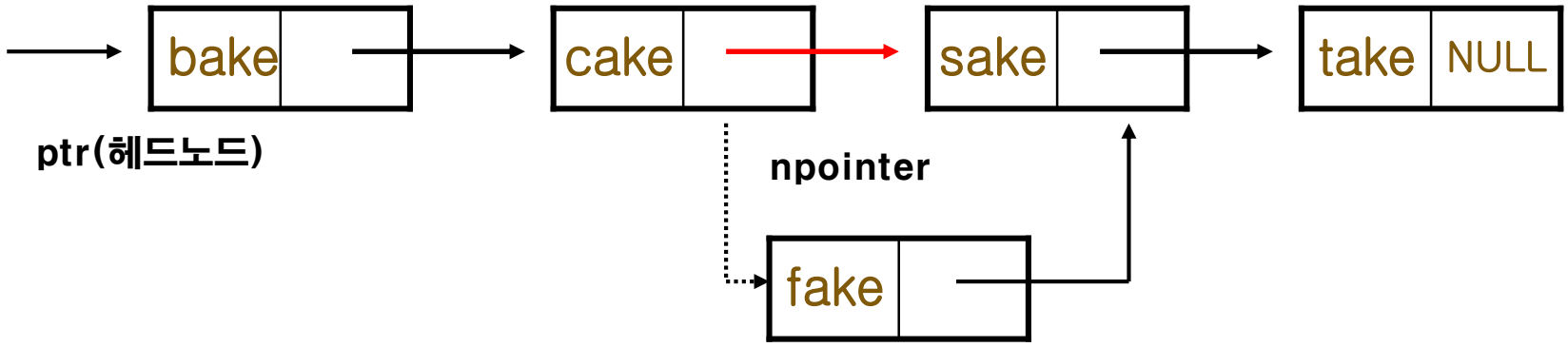
노드의 삽입

- cake뒤에 fake 삽입



노드의 삭제

- fake를 삭제할 때



단순 연결 리스트의 구현

◆ 연결 리스트를 생성하기 위해 필요한 기능

(1) 노드의 구조 정의

(2) 노드 생성 : malloc() 함수 사용

(3) 노드의 데이터 필드와 링크 필드에 값을 할당

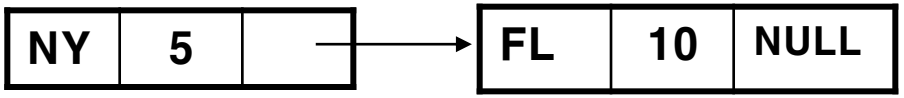
◆ [예제] ake로 끝나는 단어의 단순 연결 리스트

```
typedef struct list_node *list_pointer;
struct list_node {
        char data[5];
        list_pointer  link;
};
list_pointer ptr = NULL;      // 새로운 공백 리스트 ptr 생성
```

연결리스트를 구현한 자기참조구조체

◆ 단순 연결리스트 구성을 위한 자료구조의 예

```
typedef struct simple_list *simple_pointer;
struct simple_list {
    char state[3];
    int count;
    simple_pointer next;
};
```



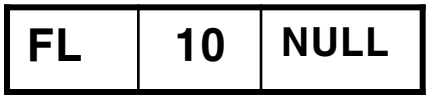
단순연결리스트 예제 I

프로그래밍 연습

[2-노드 연결리스트 만들기]

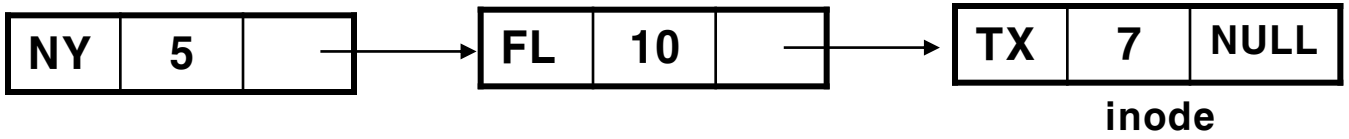
```
simple_pointer state_list()
{
    simple_pointer node1, node2;

    node1 = (simple_pointer) malloc(sizeof(struct simple_list));
    node2 = (simple_pointer) malloc(sizeof(struct simple_list));
    strcpy(node1->state, "NY");
    node1->count = 5;
    node1->next = node2;
    strcpy(node2->state, "FL");
    node2->count = 10;
    node2->next = NULL;
    return node1;
};
```

node1



node2

프로그래밍 연습



단순연결리스트 예제 II

```
strcpy(inode->state, "TX");
inode->count = 7;

while (ptr != NULL) {
    before = ptr;
    ptr = ptr -> next;
}
before -> next = inode;
inode -> next = NULL;
```

프로그래밍 연습

[마지막에 노드로 삽입]

```
void append(simple_pointer ptr, simple_pointer inode)
{
    simple_pointer before;
    while (ptr != NULL) {
        before = ptr;
        ptr = ptr -> next;
    }
    before -> next = inode;
    inode -> next = NULL;
}
```

프로그래밍 연습

[연결리스트안의 데이터 출력]

```
void print_list(simple_pointer ptr)
{
    printf("The singly linked list contains : \n");
    while (ptr != NULL) {
        printf("%s : %d\n", ptr->state, ptr->count);
        ptr = ptr -> next;
    }
}
```

프로그래밍 연습

[main()함수]

```
main()
{
    simple_pointer ptr, inode;
    ptr=state_list();
    inode = (simple_pointer) malloc(sizeof(struct simple_list));
    strcpy(inode->state, "TX");
    inode->count = 7;
    append(ptr, inode);
    print_list(ptr);
}
```

자기참조구조체를 이용하여 연결리스트 구성하기

자기참조구조체로 구성된 연결리스트를 이용한 파일 처리 프로그래밍

struct course_node 타입의 구조체를 기본 노드로 하는 연결 리스트를 기본으로 주어진 파일 안의 강좌번호(courseid), 담당교수이름(tname), 수강생 수(snum), 강의실 번호(roomnum) 를 읽어 마지막으로 자기노드와 같은 타입의 포인터, struct course_node *(즉 course_list_pointer) 인 next를 이용하여 노드를 연결하는 프로그램을 위한 자료구조

```
typedef struct course_node *course_list_pointer;
struct course_node {
    char courseid[10];
    char tname[20];
    unsigned snum;
    unsigned roomnum;
    course_list_pointer next;
};
```

자기참조구조체를 이용하여 연결리스트 구성하기

자기참조구조체로 구성된 연결리스트를 이용한 파일 처리 프로그래밍

```
main(int argc, char *argv[])
{
        FILE *coursedb;
        course_list_pointer temp, before, ptr = NULL;
        char sprof[20];
        unsigned sroomnum;

        if ((coursedb = fopen(argv[1], "r")) == NULL) {
                printf("데이터 파일을 열 수 없습니다 \n");
                exit(1);
        }
```

자기참조구조체를 이용하여 연결리스트 구성하기

자기참조구조체로 구성된 연결리스트를 이용한 파일 처리 프로그래밍

```
temp = (course_list_pointer)malloc(sizeof(struct course_node));
while (fscanf(coursedb, "%s %s %u %u", temp->courseid,
        temp->tname, &(temp->snum), &(temp->roomnum)) != EOF) {

        // 큐 방식으로 연결하기

        if (ptr)
            before -> next = temp;
        else
            ptr = temp;

        before = temp;

        temp = (course_list_pointer)malloc(sizeof(structcourse_node));
}
 before->next = NULL;
```

자기참조구조체를 이용하여 연결리스트 구성하기

자기참조구조체로 구성된 연결리스트를 이용한 파일 처리 프로그래밍

```
temp = (course_list_pointer)malloc(sizeof(struct course_node));
while (fscanf(coursedb, "%s %s %u %u", temp->courseid,
        temp->tname, &(temp->snum), &(temp->roomnum)) != EOF) {

    // 스택 방식으로 연결하기

    if (ptr)
        temp->next = ptr;
    else
        temp->next = NULL;
    ptr = temp;

    temp = (course_list_pointer)malloc(sizeof(structcourse_node));

} //while
```

자기참조구조체를 이용하여 연결리스트 구성하기

```
printf("저장된 강좌번호 : \n");
temp = ptr;
for (; temp; temp = temp->next)
    printf("%s\n", temp->courseid);
printf("=========================================\n");

printf("수강인원이 40명 이상인 강좌 코드 번호를 출력하시오.\n");
printout_over40(ptr);

printf("검색하고자 하는 교수이름 입력 : ");
scanf("%s", sprof);
printf("%s 교수가 강의하는 강좌코드와 강의실 번호를 출력하시오.\n",sprof);
search_prof(ptr, sprof);

printf("검색하고자 하는 강의실 번호 입력 : ");
scanf("%u", &sroomnum);
printf("%u 강의실에 수업하는 강좌코드와 담당교수를 출력하시오.\n",sroomnum);
search_room(ptr, sroomnum);
```

자료처리 함수 예(1)

자기참조구조체로 구성된 연결리스트를 이용한 파일 처리 프로그래밍

```
void printout_over40(course_list_pointer ptr)
{
        for (; ptr; ptr = ptr->next)
                if (ptr->snum >= 40)
                        printf("%s\n", ptr->courseid);
}
```

자료처리 함수 예(2)

자기참조구조체로 구성된 연결리스트를 이용한 파일 처리 프로그래밍

```
void search_prof(course_list_pointer ptr, char name[])
{
        for (; ptr; ptr = ptr->next)
                if (!strcmp(ptr->tname, name))
                        printf("%s\t%u\n", ptr->courseid, ptr->roomnum);
}
```

자료처리 함수 예(3)

자기참조구조체로 구성된 연결리스트를 이용한 파일 처리 프로그래밍

```
void search_room(course_list_pointer ptr, unsigned sroomnum)
{
        for (; ptr; ptr = ptr->next)
                if (ptr->roomnum == sroomnum)
                        printf("%s\t%s\n", ptr->courseid, ptr->tname);
}
```