C – 자료구조와 알고리즘

Data Structure

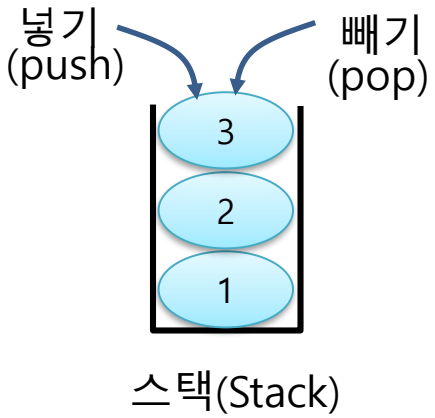
스택(Stack) 자료 구조

- **스택(Stack)**
  - 후입선출(LIFO : Last in First Out) 구조

    배열에서 나중에 들어간 자료를 먼저 꺼냄

    (응용 예: 스택 메모리, 접시 닦이, 게임 무르기)

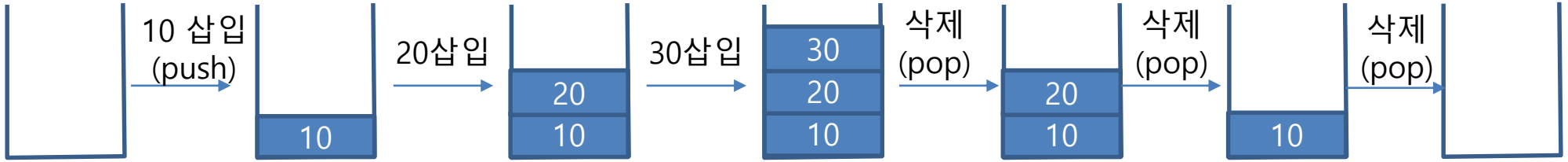
메소드명	설명
push()	원소(자료)를 스택에 넣는다.
pop()	스택의 맨 위 원소를 가져온다.



스택(Stack)

스택(Stack)

- 스택(Stack)




```c
int main()
{
    printf("스택에 자료 저장(넣기)\n");
    push(10);
    push(20);
    push(30);

    printf("스택에서 자료 삭제(빼기)\n");
    printf("%d\n", pop());
    printf("%d\n", pop());
    printf("%d\n", pop());
    printf("%d\n", pop());

    return 0;
}
```

스택(Stack)

- 스택(Stack)

```
#define MAX_LEN 10
int top = -1;
int stack[MAX_LEN];  //stack 배열 생성

void push(int x) { //요소 삽입(저장)
    top++;
    stack[top] = x;
    printf("%d %d\n", top, stack[top]);
}

int pop() { //요소 삭제(빼기)
    if (top < 0) {
        printf("스택이 비었습니다!!\n");
        return 0;
    }
    return stack[top--];
}
```

```
스택에 자료 저장(넣기)
10
20
30
스택에서 자료 삭제(빼기)
30
20
10
스택이 비었습니다!!
0
```

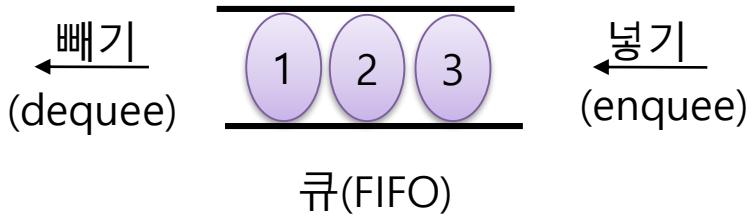
큐(Queue) 자료 구조

- **큐(Queue)**
  - 선입선출(FIFO : First in First Out) 구조

    배열에서 먼저 들어간 자료를 먼저 꺼냄

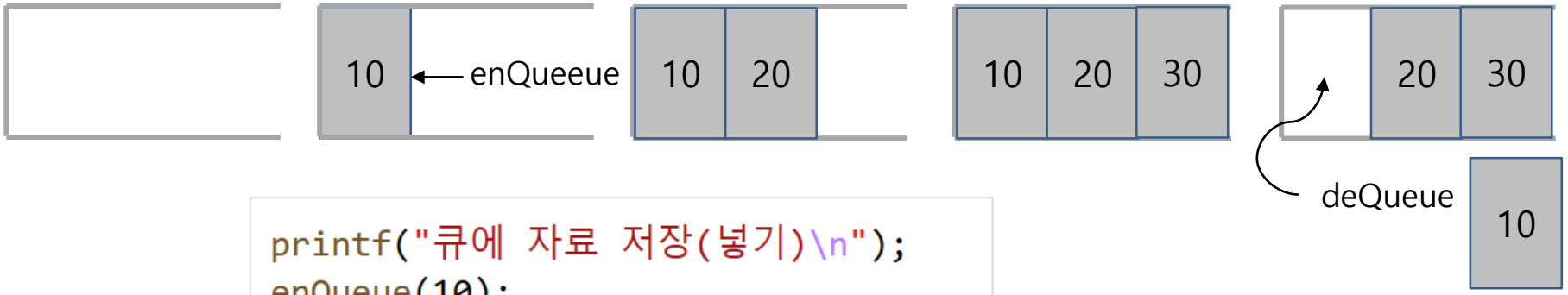
    (응용 예: 택시정류장 줄서기, 운영체제 작업큐)

메소드명	설명
enQueue()	주어진 객체를 스택에 넣는다.
dequeue()	객체 하나를 가져온다. 객체를 큐에서 제거한다.



큐(Queue)

- 큐(Queue)




```
printf("큐에 자료 저장(넣기)\n");
enQueue(10);
enQueue(20);
enQueue(30);
printf("\n");

printf("큐에서 자료 삭제(빼기)\n");
printf("%d ", deQueue());
printf("%d ", deQueue());
printf("%d ", deQueue());
printf("%d ", deQueue());
```

큐(Queue)

- 큐(Queue)

```c
#define MAX_LEN 10
int front = 0, rear = 0; //배열의 첫 인덱스, 끝 인덱스
int queue[MAX_LEN];  //queue 배열 생성

void enQueue(int x) {
    queue[rear] = x;
    printf("%d %d ", rear, queue[rear]);
    rear++;
}

int deQueue() {
    if (front == rear) {
        printf("큐가 비었습니다!!\n");
        return 0;
    }
    return queue[front++];
}
```

```
큐에 자료 저장(넣기)
10 20 30
큐에서 자료 삭제(빼기)
10 20 30 큐가 비었습니다!!
0
```

토글(toggle) 알고리즘

- 토글 알고리즘

  상태 변수를 사용하여 어떤 상태를 바꿔주면서 유지 시키는 것

  플래그 라고도 함.

```
int a[] = { 9, 8, 7, 6, 7 };
int i;
int count = 0;

//7이 몇 개인지 세기
for (i = 0; i < 5; i++){
    if (a[i] == 7) {
        printf("7 발견!\n");
        count++;
    }
}
printf("7을 %d개 발견!", count);
```

토글(toggle) 알고리즘

- 토글 알고리즘

```
//7을 하나 발견하면 종료
int sw = 0;  //상태(토글) 변수
for (i = 0; i < 7; i++) {
    if (a[i] == 7) {
        printf("7 발견!\n");
        sw = 1;
        break;
    }
}

if(sw == 0)
    printf("7을 발견 못함!\n");
```

재귀 알고리즘

- 재귀 호출

  어떤 함수 안에서 자기 자신을 부르는 것을 말한다.

  재귀 호출은 무한 반복하므로 종료 조건이 필요하다.

```
void func(int n)
{
    printf("Help Me!\n");
    n--;
    if (n > 0)  //종료 조건
        func(n);
    /*
      n=4, func(4), Help Me!
      n=3, func(3), Help Me!
      n=2, func(2), Help Me!
      n=1, func(1), Help Me!
      n=0, 반복 종료
    */
}
```

```
int main()
{
    func(4);

    return 0;
}
```

재귀 알고리즘

- 팩토리얼 계산하기

```
int factorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n * factorial(n - 1);
}
```

재귀 알고리즘

- 팩토리얼 계산하기

```
int main()
{
    int a, b, c;

    a = factorial(1);  // 1 * factorial(0), 1 * 1 = 1
    b = factorial(2);  // 2 * factorial(1), 2 * 1 = 2
    c = factorial(3);  // 3 * factorial(2), 3 * 2 = 6

    printf("1!=%d, 2!=%d, 3!=%d\n", a, b, c);

    return 0;
}
```

재귀 알고리즘

- 십진수를 이진수로 변환하기

```
int printBin(int a)
{
    if (a == 0 || a == 1) printf("%d", a);
    else
    {
        printBin(a/2);
        printf("%d", a%2);
    }
}

int main()
{
    int x = 11;
    printBin(x);

    return 0;
}
```

정렬 알고리즘

- 단순 정렬(Bubble Sort)

```c
//단순 정렬 - 내림차순
int a[5] = { 3, 2, 5, 1, 4 };
int i, j, temp;

for (i = 0; i < 4; i++) {
    for (j = i + 1; j < 5; j++) {
        if (a[i] < a[j]) { //'>'면 오름차순
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }
}

for (i = 0; i < 5; i++) {
    printf("%-2d", a[i]);
}
```

```c
/*
  i=0, j=1, 3<2, 교환없음
       j=2, 3<5, 5,2,3,1,4
       j=3, 3<1, 교환없음
       j=4, 3<4, 5,2,4,1,3
  i=1, j=2, 2<4, 5,4,2,1,3
       j=3, 2<1, 교환없음
       j=4, 2<3, 5,4,3,1,2
  i=2, j=3, 3<1  교환없음
       j=4, 3<2  교환없음
  i=3, j=4, 1<2  5,4,3,2,1
*/
```

정렬 알고리즘

- 버블 정렬(Bubble Sort)

```c
//버블 정렬 - 오름차순
int a[5] = { 3, 2, 5, 1, 4 };
int i, j, temp;

for (i = 0; i < 5; i++) {
    for (j = 0; j < 4; j++) {
        if (a[j] > a[j+1]) { //'<'면 내림차순
            temp = a[j];
            a[j] = a[j+1];
            a[j+1] = temp;
        }
    }
}

for (i = 0; i < 5; i++) {
    printf("%-2d", a[i]);
}
```

```c
/*
  i=0, j=0, 3>2, 2,3,5,1,4
       j=1, 3>5, 교환없음
       j=2, 5>1, 2,3,1,5,4
       j=3, 5>4, 2,3,1,4,5
  i=1, j=0, 2>3, 교환없음
       j=1, 3>1, 2,1,3,4,5
       j=2, 1>4, 교환없음
       j=3, 4>5, 교환없음
  i=2, j=0, 2>1, 1,2,3,4,5 //최종 저장
  i=3, 교환없음
  i=4, 교환없음
  i=5, 반복종료
*/
```

검색 알고리즘

- 순차 검색(Sequential Search)

  처음부터 끝까지 순서대로 찾음

```c
int a[9] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
int i; //인덱스
int x = 8;  //찾을 값
int found = 0;  //상태 변수(찾음, 못찾음)

printf("===== 순차 검색 =====\n");
for (i = 0; i < 9; i++) {
    if (a[i] == x) {
        printf("%d은 a[%d]에 있습니다.\n", x, i);
        found = 1;  //찾음
        break;
    }
}
if (!found) { //못찾음
    printf("%d는 없습니다.\n", x);
}
```

검색 알고리즘

- 이분 검색(Binary Search)

  정렬된 데이터를 좌우 둘로 나눠서 찾는 값의 검색 범위를 좁혀가는 방법이다.

  - 찾을 값 < 가운데 요소 -> 오른쪽 반을 검색 범위에서 제외시킴
  - 찾을 값 > 가운데 요소 -> 왼쪽 반을 검색 범위에서 제외시킴
  - 찾을값 = 가운데 요소 -> 검색을 완료함

검색 알고리즘

- 이분 검색(Binary Search)

```
printf("===== 이분 검색 =====\n");
int low, high, mid;
int x;  //검색할 값
int found; //상태 플래그

int a[9] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

low = 0;  //첫 인덱스
high = 8; //마지막 인덱스
x = 8;
found = 0;

while (low <= high) {
    mid = (low + high) / 2; //중간값
    if (a[mid] == x) {
        found = 1;
        break;
    }
```

검색 알고리즘

- 이분 검색(Binary Search)

```
        else if (a[mid] < x) {
            low = mid + 1;
        }
        else {
            high = mid - 1;
        }
        /*
            a[4] < 8, low=5, mid=6
            a[6] < 8, low=7, mid=7
            a[7] = 8  찾음
        */
    }
    if (a[mid] == x) {
        printf("%d은 a[%d]에 있습니다.\n", x, mid);
    }
    else {
        printf("%d는 없습니다.\n", x);
    }
```

연결 리스트(Linked List)

- 연결 리스트(Linked List)의 필요성

```
/*
    - 배열 자료구조의 문제 및 연결 리스트의 필요성
    배열에서 특정 요소를 삭제하면
    빈 방을 왼쪽으로 이동하지 않고 놔둔다면 새로운 데이터를 넣을때마다
    빈 방을 찾아야하는 번거로움이 있다.
    배열은 이러한 메모리 관리가 필요하고 이러한 문제를 해결하는 것이
    연결 리스트이다.
*/

//Student 구조체 정의
typedef struct {
    int number;
}Student;
```

연결 리스트

- 연결 리스트(Linked List)의 필요성

```
int main()
{
    Student s[10]; //구조체 배열 생성
    int i;

    for (i = 0; i < 10; i++) {
        s[i].number = i + 1; //요소 저장
    }

    for (i = 0; i < 10; i++) { //전체 출력
        printf("%d ", s[i].number);
    }
    printf("\n");

    printf("2번 학생 전학\n");
    s[1].number = 0;  //요소 삭제
```

연결 리스트

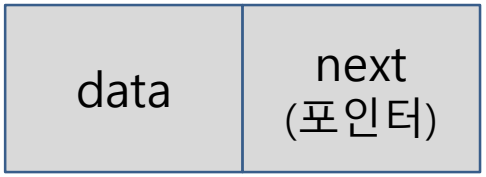
- 연결 리스트(Linked List)의 필요성

```
    for (i = 0; i < 10; i++) {
        printf("%d ", s[i].number);
    } //1 0 3 4 5 6 7 8 9 10
    printf("\n");

    for (i = 1; i < 9; i++) {
        s[i].number = s[i + 1].number;
    }
    s[9].number = 0;

    printf("방을 왼쪽으로 이동\n");
    for (i = 0; i < 10; i++) {
        printf("%d ", s[i].number);
    }//1 3 4 5 6 7 8 9 10 0

    return 0;
}
```

연결 리스트

- 연결 리스트(Linked List)의 필요성
  - 링크드 리스트의 각 요소는 다음 요소를 가리키는 주소 값을 가지고 물리적인 메모리는 떨어져 있더라도 논리적으로는 앞뒤 순서가 있다.
  - 배열은 자료를 삭제시 공간을 비워서 뒤 요소를 밀고 그 자리에 놓지만, 링크드 리스트는 주소 값만 변경하여 자료 이동이 발생하지 않음
  - 노드(node) – 데이터와 주소를 가지는 단위

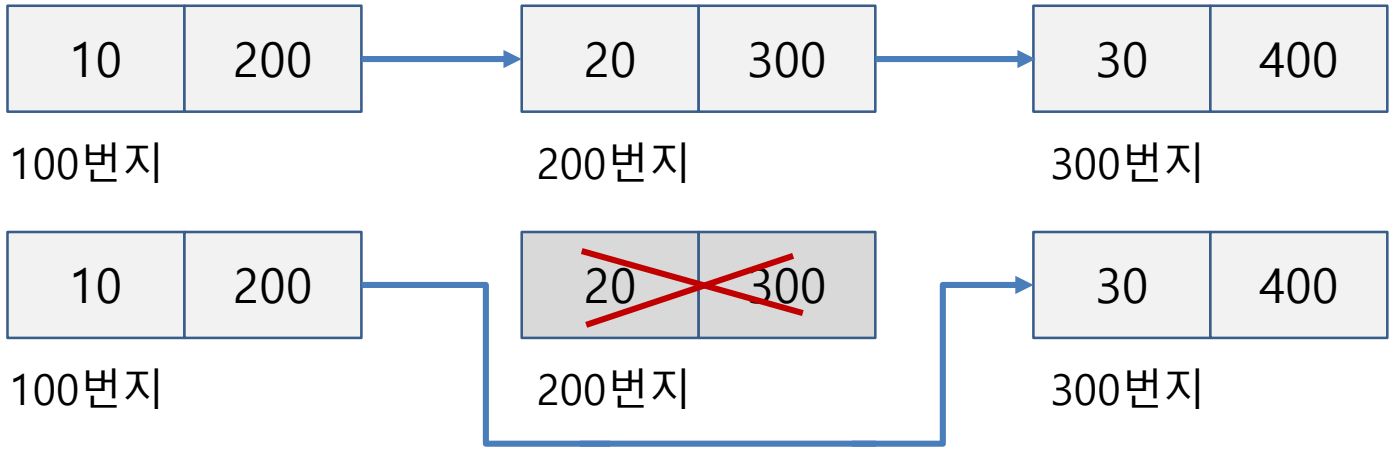
노드(Node)



- data: 노드가 가지는 데이터 값
- next: 포인터, 다음 노드의 주소 저장

연결 리스트

- 구조체를 활용한 연결 리스트(Linked List) 구현




```c
/*
    - 자기 참조 구조체: struct 내부에 동일한 타입의 포인터 멤버 포함
    - 포인터를 이용한 노드 연결과 순회
*/
typedef struct {
    int data;
    struct List* next; //자기 참조 구조체
}List;
```

연결 리스트

- 구조체를 활용한 연결 리스트(Linked List) 구현

```
List x, y, z; //노드 생성 및 초기화

x.data = 10;
y.data = 20;
z.data = 30;

//노드 연결(Linked list 구성)
x.next = &y;   //x -> y
y.next = &z;   //y -> z
z.next = NULL; //z는 마지막 노드(리스트 끝)

//리스트 순회 및 출력
List* p;  //포인터 이용
p = &x;   //초기화

printf("%d %x\n", x.data, p->next);  //첫번째 노드 출력

p = p->next;
printf("%d %x\n", y.data, p->next);  //두번째 노드
```

연결 리스트

- 구조체를 활용한 연결 리스트(Linked List) 구현

```
//전체 출력
for (p = &x; p != NULL; p = p->next) {
    printf("%d ", p->data);
} //10 20 30

//노드 삭제(y 제거)
printf("\n구조체 y 삭제 후\n");
x.next = y.next; // x -> z
y.next = NULL;   // y는 연결에서 제외

for (p = &x; p != NULL; p = p->next) {
    printf("%d ", p->data);
} //10 30
```

```
10 acaffc48
20 acaffc78
10 20 30
구조체 y 삭제 후
10 30
```

연결 리스트

- 구조체를 활용한 연결 리스트(Linked List) 구현

```
//노드 생성 및 초기화
Person a = {8, "이강인"};
Person b = {6, "이정후"};
Person c = {7, "최민정"};
Person d = {7, "신유빈"};

a.next = &b;  //구조체 a 뒤에 구조체 b 연결
b.next = &c;
c.next = &d;
d.next = NULL;

Person* p;  //구조체 포인터 변수 선언
p = &a;     //초기화

//첫 번째 노드 출력
printf("%d %s\n", p->age, p->name); //8 이강인
```

연결 리스트

- 구조체를 활용한 연결 리스트(Linked List) 구현

```
//노드 순회 및 출력
for (p = &a; p != NULL; p = p->next)
{
    printf("%d ", p->age);
    printf("%s\n", p->name);
}

printf("= 구조체 c 삭제 후 =\n");
b.next = c.next; //b -> d
c.next = NULL;

for (p = &a; p != NULL; p = p->next)
{
    printf("%d ", p->age);
    printf("%s\n", p->name);
}
```