포인터의 개념

- 포인터는 메모리 주소를 값으로 가지는 데이터 형(type)이다.
- C 언어에서는 어떤 타입 T에 대해서 T의 포인터 타입이 존재한다.

  int * float *
- 포인터 타입에는 주소연산자(&)와 역참조(간접 지시) 연산자(*)가 사용된다.

[예제1]

```
main()
{
    int *p, q;
    q = 100;
    p = &q;
    printf("%d", *p);
}
```

포인터의 개념

- 메모를 주소를 값으로 가지는 데이터 형

[예제2]

```
main()
{
    int *p, q;
    float *fp, x;
    p = &q;
    *p = 199;
    scanf("%f", fp);
    x = *fp;
    printf("%d --- %.2f\n", q, x);
}
```

```
fp = &x;
scanf("%f", fp);
```

포인터의 활용

[예제3] x=10, y=30, z=5 의 값이 입력되었을 때를 예로 하여 다음 프로그램을 수행시켜 보자.

```
main()
{
    int x, y, z;
    printf("세 수를 입력하시오.  “);
    scanf("%d %d %d", &x, &y, &z);
    if (x > y) swap(&x, &y);
    if (y > z) swap(&y, &z);
    if (x > y) swap(&x, &y);
    printf("%d **** %d **** %d\n", x, y, z);
}
```

```
void swap(int *px, int *py)
{
    int temp;
    temp = *px;
   *px = *py;
   *py = temp;
}
```

포인터의 활용

- 함수 호출시 파라메터로서 일반 변수를 사용하는 경우와 포인터 변수를 사용하는 경우의 대표적인 예

```
printf("%d", num);
scanf("%d", &num)
```

함수 호출시 파라메터로서

- 일반 변수를 사용하는 경우 : 함수에서 사용할 변수의 값을 설정하여 보내 줌.
- 포인터 변수를 사용하는 경우 : 함수를 수행하고 복귀한 후 파라메터 변수 값이 변화하는 경우

포인터의 활용

[문제] 다음 함수 add_product()를 수행한 후 main()의 printf문에서 출력되는 변수 x, y, z의 값을 쓰시오.

```
main()
{
    int x = 10, y = 20, z;
    z = add_product(&x, y);
    printf("x = %d :: y = %d :: z = %d\n", x, y, z);
}
```

```
int add_product(int *a, int b)
{
    int temp=b;
    b = *a + b;
    *a = *a * temp;
    return b;
}
```

x = 200 :: y = 20 :: z = 30

포인터의 활용

[예제4] 포인터변수가 가리키는 데이터 공간을 실행시간에 할당 받기 위하여 malloc 함수 사용

(type *) malloc(sizeof(type))

```
main()
{
    int *ip;
    float *fp;
    ip = (int *) malloc(sizeof(int));
    fp = (float *) malloc(sizeof(float));
    *ip = 2008;      *fp = 7.123;
    printf("year = %d : point = %.3f\n", *ip, *fp);
    free(ip);  free(fp);
}
```

포인터와 배열

```
#define MAXDATA 100
float diff(float *a, int n);
main()
{
    float xarray[MAXDATA];
    int n,k;
    printf( “Input the number of data to be processed : );
    scanf( “%d” , &n);
    printf("Enter %d reals :  “, n);
    for (k=0; k<10; k++)
        scanf("%f", xarray+k);
    printf("The rage of data value = %.2f\n", diff(xarray, n));
}
```

- 배열의 이름은 배열 첫 데이터의 주소이고 배열 전체의 대표 정보이다.
- C언어에서는 배열을 함수의 파라미터로 넘겨 줄 때 배열의 이름을 전달한다.

```
for (k=0; k<10; k++)
    scanf("%f", &xarray[k]);
```

&xarray[0]

포인터와 배열

```
float diff(float a[], int n)
{
        float max = a[0], min=a[0];
        int i;
        for (i = 1; i<n; i++) {
                if (a[i] > max) max = a[i];
                if (a[i] < min) min = a[i];
        }
        return max - min;
}
```

```
float diff(float *a, int n)
```

연결리스트의 개념

순차리스트 구조에서 연결리스트 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 생성하여 연결해서 사용하는 구조가 적합하다.

연결리스트의 개념

◆ 연결 리스트의 구조

- [데이터, 링크]의 형태의 노드를 기본 단위로 연결되어 있음

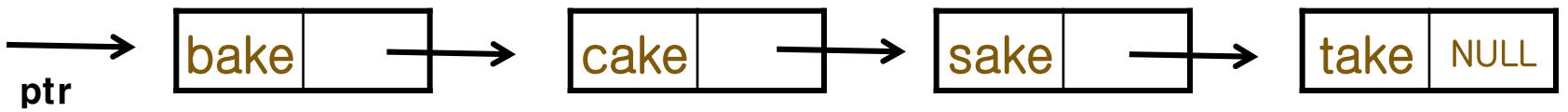
데이터	링크

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

단순 연결 리스트 (singly linked list)

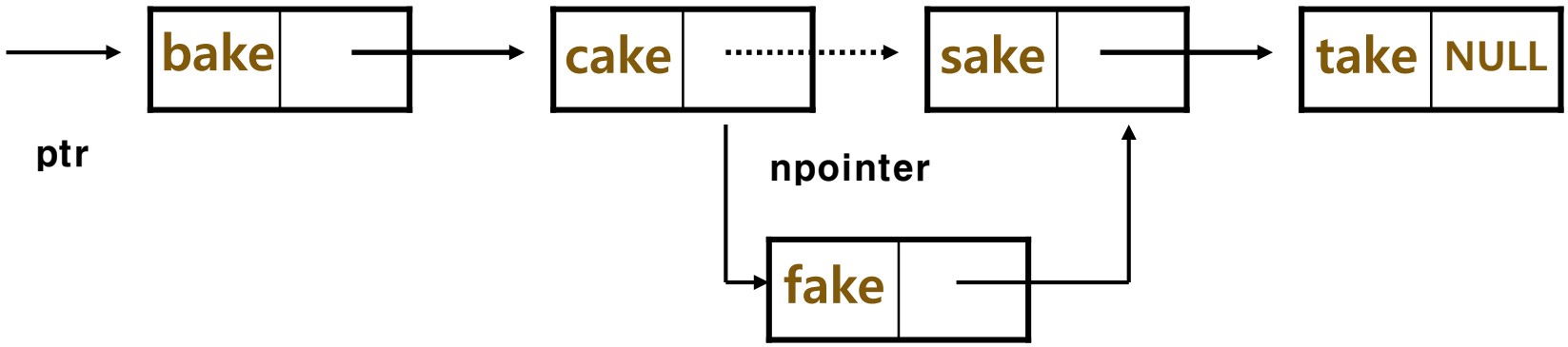
◆ 단순 연결리스트의 개념

- 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



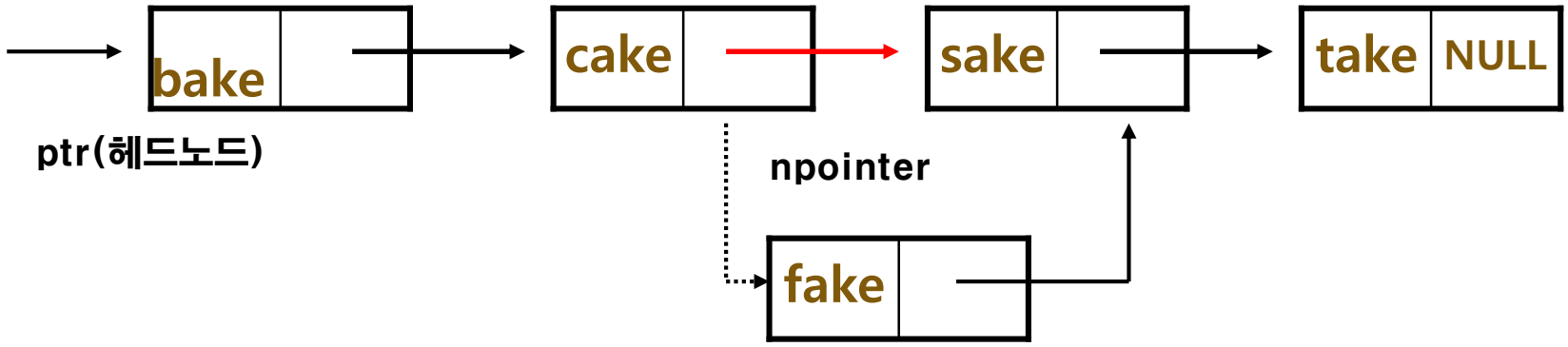
노드의 삽입

- 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

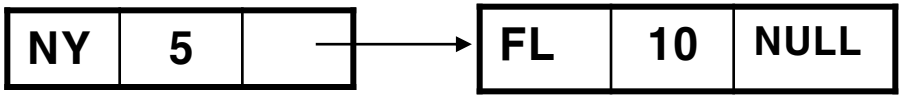
◆ [예제] ake로 끝나는 단어의 단순 연결 리스트

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

프로그래밍 연습

◆ 단순 연결리스트 프로그래밍 연습

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



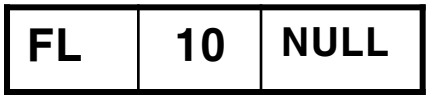
단순연결리스트 예제 I

프로그래밍 연습

[2-노드 연결리스트 만들기]

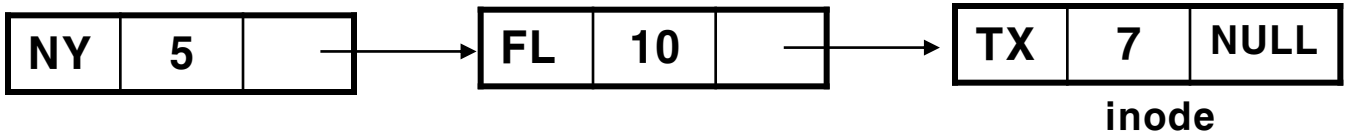
```
simple_pointer state_list()
{
    simple_pointer node1, node2;

    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 실습

◆ 자료구조 정의

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

◆ 함수 선언

```
simple_pointer state_list();
void append(simple_pointer ptr, simple_pointer inode);
void print_list(simple_pointer ptr);
```

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);
}
```