프로그래밍 기초

16장. 동적 메모리 Part 2

학습목표

- 동적 메모리 할당 방식을 이해하고 설명할 수 있다.
  - 정적 할당과 동적 할당의 필요성
  - 함수 malloc(), calloc(), realloc(), free()의 사용
  - 동적 메모리 할당으로 구현하는 자기참조 구조체

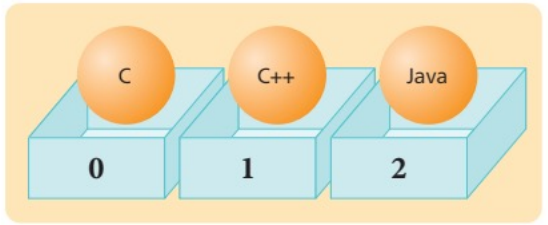
- **연결 리스트를 이해하고 설명할 수 있다.**
  - 연결 리스트의 이해와 구현
  - 연결 리스트의 노드, 헤드, 순회, 삽입, 삭제
  - 여러 파일로 구성하는 프로젝트와 사용자 정의 헤더파일
  - 연결 리스트의 구현

- 전처리 지시자를 이해하고 설명할 수 있다.
  - 매크로와 인자를 활용한 매크로
  - #if와 #endif
  - #ifdef와 #endif, #ifndef #endif
  - 전처리 연산자 #, #@, ##, defined

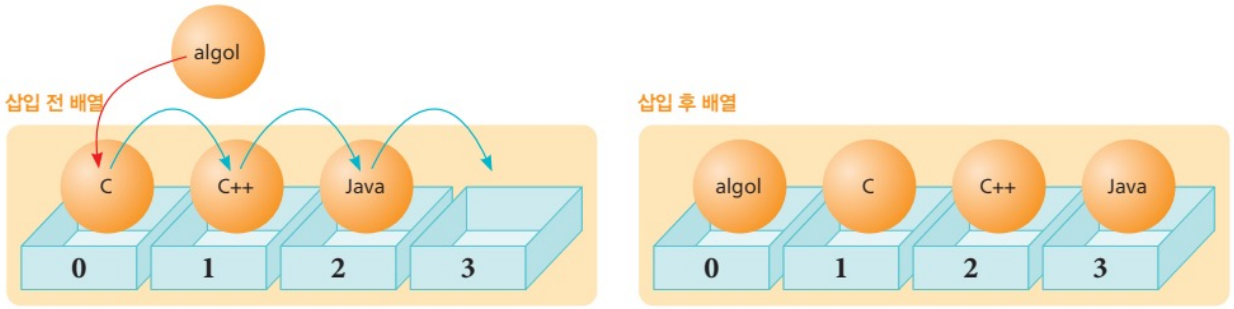
배열의 장단점

- **프로그램 언어를 개발된 순서인 C, C++, Java 순서로 나열 방법**
  - 배열

1	C
2	C++
3	Java

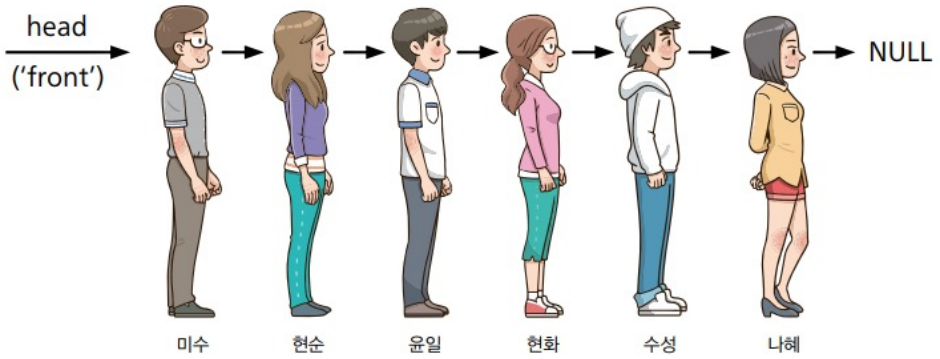


- **자료를 순차적으로 저장하기 가장 쉬운 방법: 배열**
  - 배열 이름과 첨자(index)를 사용, 원하는 원소를 직접 임의 참조(random access) 가능
  - 단점
    - 컴파일 전에 배열의 크기가 이미 결정: 실행 중간에 배열 크기를 변경하는 것이 불가능
    - 맨 앞이나 중간에 새로운 자료를 삽입: 삽입되는 자료 이후의 원소가 모두 이동
    - 중간에 하나 삭제하는 경우도 마찬가지 어려움



연결 리스트 개요

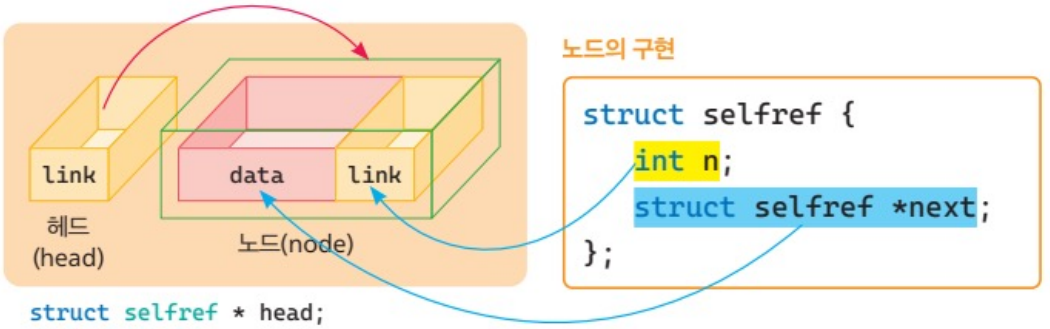
- **연결 리스트(Linked List)**
  - 배열과 함께 순차적 자료 표현에 적합한 구조
  - 헤드(head)에서 시작하여 가리키는 곳을 따라가면서 순차적으로 자료를 표현
  - 원소인 노드(node)가 순차적으로 연결된 자료구조
    - 노드는 배열의 원소에 해당: 자료(data)와 링크(link)로 구성
    - 노드는 자기 참조 구조체로 정의
  - 인덱스 대신 링크(link)라는 포인터로 다음 노드를 가리키는 구조

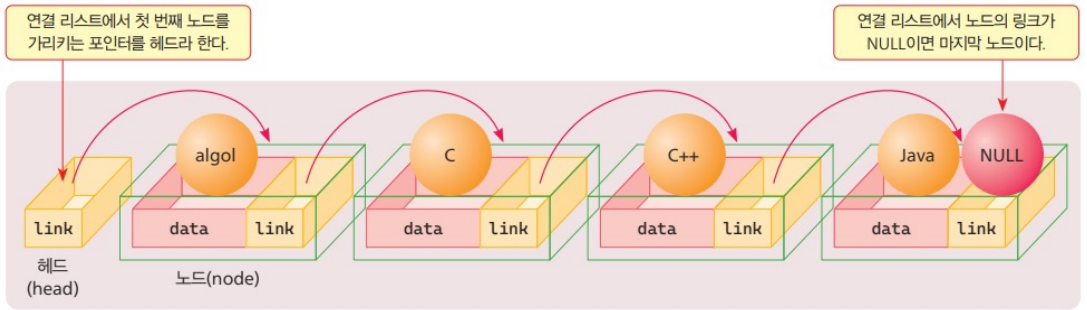


- **연결 리스트 예**
  - 헤드(head)는 "미수"를 가리키고
    - "미수"는 다시 "현순"을 가리키고, 계속해서 "윤원", "현화", "수성", "나혜"
    - 그리고 나혜는 마지막 순서로 가리키는 사람이 없는 경우 NULL 값을 가짐

노드의 표현

- **노드의 자료: 필요한 여러 변수의 조합으로 구성**
  - 노드의 링크: 자기 구조체의 포인터로 구현
- **헤드(head)**
  - 첫 번째 노드를 가리키는 포인터
- **테일(tail)**
  - 마지막 노드를 가리키는 포인터
- **연결 리스트로 표현한 그림**
  - 헤드 포인터 노드에서 시작해서
    - ➢ 화살표(링크)를 따라 이동하면서 자료를 순서대로 참조
  - 연결 리스트에서 마지막 노드의 링크는 NULL로 저장
    - ➢ 만일 연결 리스트에 노드가 하나도 없다면 헤드는 NULL





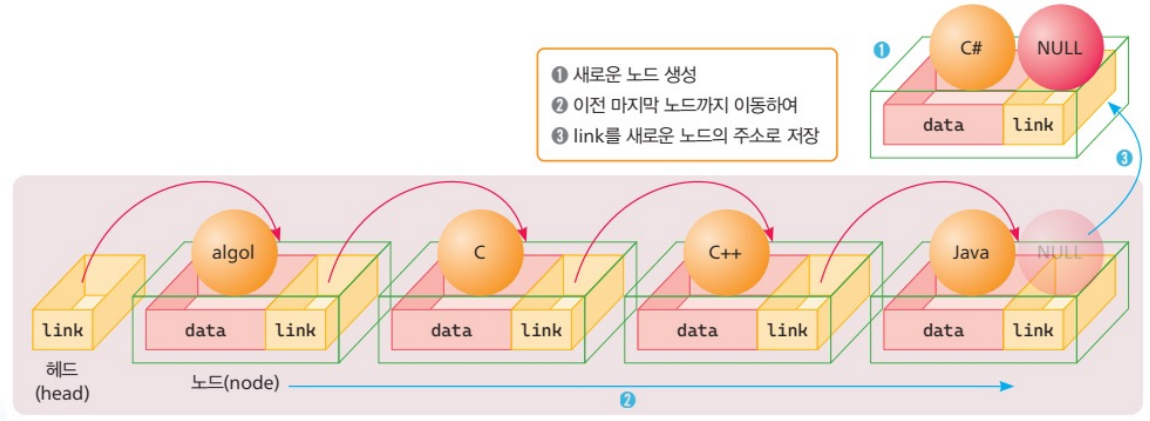
연결 리스트 장단점

- **연결 리스트 장점**
  - 프로그램 내부에서 메모리가 허용하는 범위에서 항목 수를 늘릴 수 있음
    - 배열과는 달리 프로그램 실행 전에 미리 기억 장소를 확보해 둘 필요가 없음
  - 프로그램 실행 중이라도 필요할 때 노드를 동적으로 생성
    - 기존의 연결 리스트에 삽입 또는 추가 가능
  - 기억 장소를 비순차적으로 사용
    - 연결 리스트에서 중간에 노드를 삽입 또는 삭제하더라도 배열에 비하여 다른 노드에 영향을 덜 미침
  - 결론
    - 연결 리스트는 동적으로 노드를 생성
    - 리스트 크기의 증가 감소에 따라 효율적으로 대처할 수 있음
    - 노드의 삽입과 삭제와 같은 자료의 재배치를 빠르게 처리

- **단점**
  - 배열에 비하여 임의 접근(random access)에 많은 시간이 소용
  - 노드 검색은 헤드에서부터 링크를 따라가는 순차적 검색만이 가능

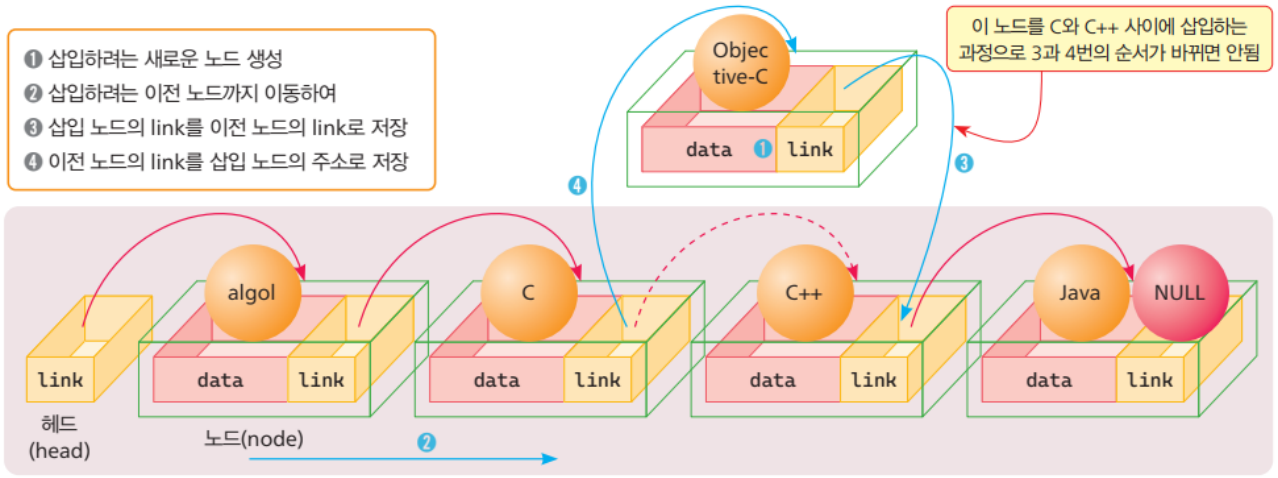
노드 순회와 추가

- **노드 순회(node traversal)**
  - 연결 리스트에서 모든 노드를 순서대로 참조하는 방법
    - 헤드부터 계속 노드 링크가 가리키는 주소(포인터)로 이동
      - 링크가 NULL이면 마지막 노드
      - 각 노드의 자료를 참조, 원하면 출력도 가능
- **노드 추가**
  - 새로운 노드를 하나 생성, 연결 리스트의 마지막 노드로 추가
  - "C#" 노드를 만들어 기존의 연결 리스트에 추가하는 방법
    - ❶ 첫 번째로 추가할 노드를 먼저 생성(malloc())후, 자료를 저장하고 링크를 NULL로 저장
    - ❷ 기존 연결 리스트를 순회하여 마지막으로 이동(Java)
    - ❸ 마지막 노드의 링크를 새로 생성한 노드의 주소로 저장하여 연결 리스트의 마지막 노드로 연결



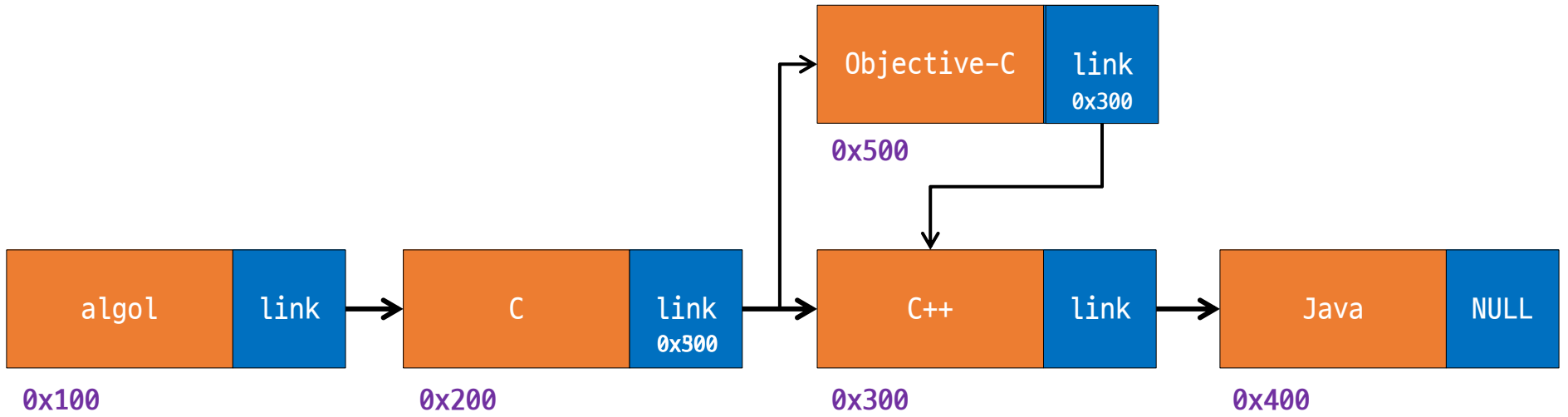
노드 삽입

- **연결 리스트 중간에 한 노드를 삽입하는 과정**
  - 기존의 연결 리스트인 노드 "C"와 노드 "C++" 사이에 노드 "Objective-C"를 삽입하는 과정
    - ➊ 삽입할 노드를 동적으로 생성하여 적당한 자료를 저장("Object-C", link=NULL)
    - ➋ 삽입하려는 바로 이전 노드인 노드 "C"로 이동
    - ➌ 삽입하는 "Objective-C" 노드의 link에 노드 "C"의 link 주소를 저장(기존: C의 link는 C++을 가리킴)
    - ➍ 노드 "C"의 link에는 "Objective-C" 노드의 주소를 저장



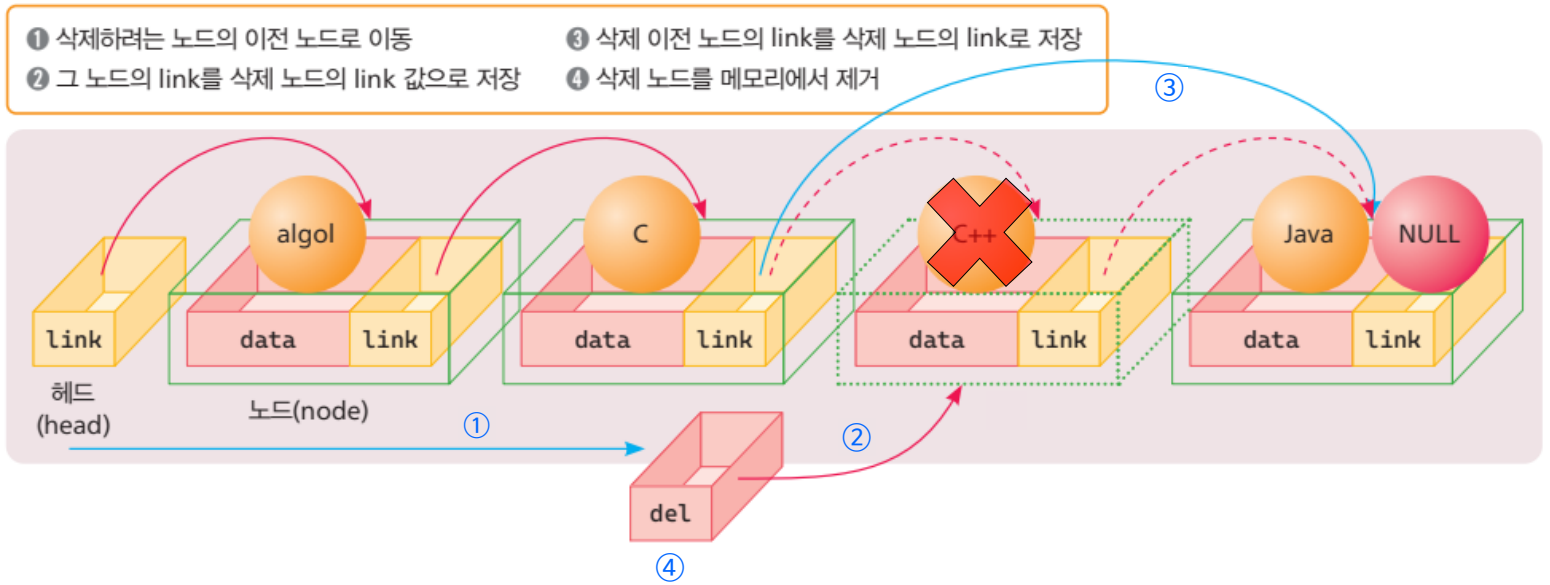
- **노드 삽입의 다른 조건**
  - 삽입 노드의 위치를 알려주는 방법
    - 위 과정 중에서 2번의 이동이 없이 바로 노드 "C"를 알려주고 노드 "C"의 다음에 삽입
    - 위 과정 중 2번을 제외하고 처리하면 새로운 노드를 삽입 가능

노드 삽입


Objective-C
link
0x300
0x500
algol
link
C
link
0x300
C++
link
Java
NULL
0x100
0x200
0x300
0x400

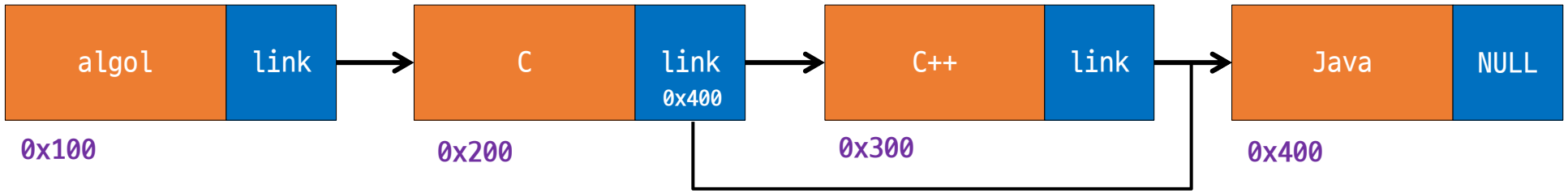
노드 삭제

- **연결 리스트에서 노드 하나를 삭제하는 과정**
  - 노드 "C++"를 삭제하려면
    - ❶ 삭제하려는 노드 바로 이전 노드("C")로 이동
    - ❷ 삭제하려는 노드("C++")의 주소를 포인터 변수(del)에 저장
    - ❸ 노드("C")의 link를 삭제하려는 노드 "C++"의 link 노드 값으로 저장
      - C++의 link 노드의 값: Java 노드의 주소
    - ❹ 포인터 변수 del로 삭제할 노드 "C++"를 메모리에서 제거



노드 삭제 과정

- C++ 노드 삭제



구조체 정의와 생성

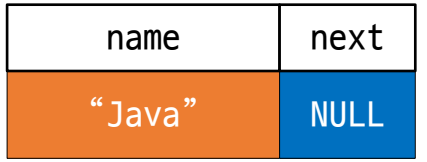
- **연결 리스트의 노드를 표현하는 구조체 struct linked_list를 정의**
  - 구조체 struct linked_list의 멤버
    - 문자열을 저장할 char *name
    - 그리고 다른 구조체를 가리킬 포인터 struct linked_list *next를 구성
  - 문장 typedef를 이용
    - struct linked_list를 간단히 NODE로 정의
    - 이 NODE의 포인터를 LINK로 정의

```
struct linked_list {              // 자기 참조 구조체 정의
    char* name;
    struct linked_list* next;
};

typedef struct linked_list NODE; // struct linked_list를 NODE로 재정의
typedef NODE* LINK;              // NODE *를 LINK로 재정의
```

구조체 정의와 생성

- **함수 createNode()**
  - 구조체 NODE를 하나 생성하고 인자로 전달되는 문자열을 구조체 멤버 name에 저장
    - ➢ 노드 하나를 생성해서 "Java" 문자열을 저장: 함수 호출 createNode("Java")로 수행

- **구현**
  - 변수 cur가 가리키는 구조체의 멤버 name은 문자 포인터: char *name
  - cur->name에 저장할 문자의 수만큼 메모리를 할당
    - ➢ 함수 malloc()에서 할당할 메모리의 크기
      - – strlen(str)+1개의 문자 크기를 확보
  - 함수 strcpy(cur->name, name)를 이용
    - ➢ "Java"가 저장된 문자열 name을 cur->name에 복사
  - 현재 생성된 노드의 멤버 cur->next
    - ➢ 초기에 NULL을 입력
    - ➢ 필요하면 나중에 다른 구조체를 가리키도록 대입




```
LINK createNode(char* name) //노드를 생성하는 함수
{
   LINK cur; //새로 생성되는 노드의 주소를 저장할 변수 cur를 선언
   cur = (LINK)malloc(sizeof(NODE));
   if (cur == NULL)
   {
      printf("노드 생성을 위한 메모리 할당에 문제가 있습니다.\n");
      return NULL;
   }
   //언어 이름을 저장할 문자배열을 동적 할당하여 name에 저장
   cur->name = (char*)malloc(sizeof(char) * (strlen(name) + 1));
   strcpy(cur->name, name);
   cur->next = NULL; //다음 노드는 모르므로 NULL로 저장

   return cur;
}
```

구조체 노드 추가(1)

- **연결 리스트에 구조체 포인터(LINK cur)를 추가하는 함수 append()**
  - cur 노드를 연결 리스트 head의 마지막 노드에 추가하는 함수
  - 인자: 시작 노드를 가리키는 LINK head, 추가될 노드인 LINK cur
  - 동작 과정
    - 연결 리스트는 인자인 노드 head가 가리키는 노드에서 시작
    - next의 주소가 NULL일 때까지 연결 리스트를 이동: 연결 리스트의 마지막 노드를 찾는 과정

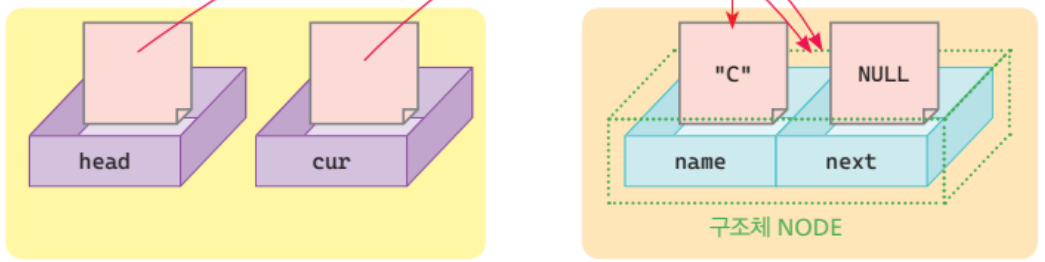
```
LINK append(LINK head, LINK cur)
{
    LINK nextNode = head;
    if (head == NULL)
    {
        head = cur; //추가하려는 노드가 head가 됨
        return head;
    }
    //멤버 next가 NULL일 때까지 이동하여 마지막 노드까지 이동
    while (nextNode->next != NULL)
        nextNode = nextNode->next;

    nextNode->next = cur; //추가 노드를 현재 노드의 next에 저장
    return head;
}
```

구조체 노드 추가(2)

- **함수 append()를 이용하여 노드가 하나도 없는 연결 리스트에 노드 C를 추가하는 과정**
  - 연결 리스트의 마지막 노드를 찾아가는데 필요한 지역 변수 nextNode를 선언
    - ➢ 초기값으로 head를 저장
  - 만일 현재 연결 리스트에 아무 노드가 없는 상태라면
    - ➢ 새로 추가되는 노드(LINK cur)가 첫 번째 노드가 됨
      - head에 cur를 대입: head가 첫 노드("C")를 가리킴
      - head를 반환하면서 함수를 종료

```
//지역 변수 nextNode를 선언하고 초기값으로 head를 저장
LINK nextNode = head;
//만일 현재 헤드가 가리키는 것이 없다면, 즉 연결리스트의 노드가 하나도 없는 경우
if (head == NULL)
{
    head = cur; //추가하려는 노드가 head가 됨
    return head;
}
```

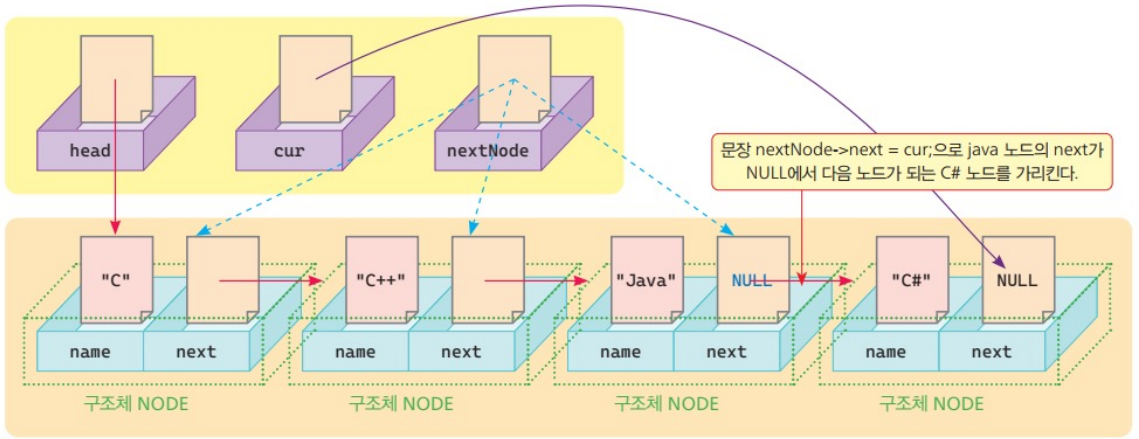
실제 name은 문자 포인터이므로 문자열 "C"가 저장된 저장 공간의 주소값을 갖는다. 편의를 위해 이와 같이 간단히 표현한다.



구조체 노드 추가(3): 연결 리스트 마지막 노드에 추가하는 과정

- **연결 리스트 마지막에 노드 C#을 추가하는 과정**
  - while() 문을 이용하여 nextNode는 멤버 next의 값이 NULL인 마지막 노드로 이동
    - ➢ nextNode는 처음에 head에서부터 마지막 노드를 찾아갈 때까지 계속 다음 노드를 가리키는 노드
      - – 결국 멤버 next의 값이 NULL인 마지막 노드를 가리킴
  - 마지막 노드에 새로운 노드 C#을 연결
    - ➢ 마지막 모드를 가리키는 nextNode를 이용하여 nextNode->next에 cur를 대입

```
//멤버 next가 NULL일 때까지 이동하여 마지막 노드까지 이동
while (nextNode->next != NULL)
{
   nextNode = nextNode->next;
}
nextNode->next = cur;  //추가 노드를 현재 노드의 next에 저장
```



구조체 노드 출력

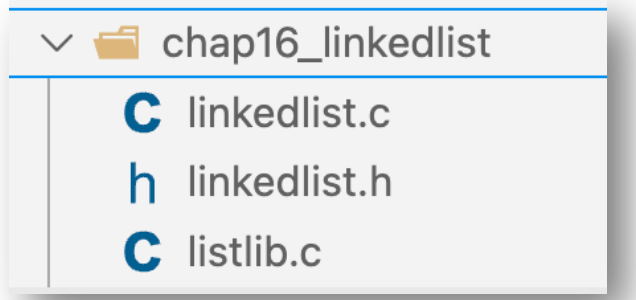
- **연결 리스트 자료 출력 함수 printList()**
  - 연결 리스트의 시작 노드를 가리키는 head 노드를 인자로 받아서 연결 리스트를 순회
    - ➢ 모든 노드의 자료를 출력하는 함수
  - 출력된 노드의 수를 반환

```
//연결 리스트의 모든 노드 출력 함수
int printList(LINK head)
{
    int cnt = 0; //방문한 노드의 수를 저장
    LINK nextNode = head;

    // nextNode를 이용하여 연결 리스트의 처음부터 끝까지 순회
    while (nextNode != NULL)
    {
        //리스트의 노드를 방문하여 방문 횟수와 문자열 자료를 출력
        printf("%3d번째 노드는 %s\n", ++cnt, nextNode->name);
        nextNode = nextNode->next;
    }
    return cnt; // 노드 방문 횟수를 반환하고 함수를 종료
}
```

프로그래밍 언어 종류를 연결 리스트로 구현

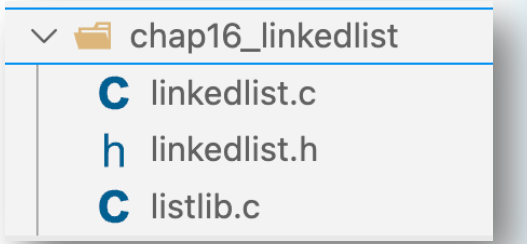
- **프로그래밍 언어 이름을 표준입력으로 받아 계속 연결 리스트에 추가**
  - 2 개의 소스 파일
    - linkedlist.c, listlib.c
  - 하나의 헤더파일
    - linkedlist.h로 구성
  - 큰 규모의 프로그램을 작성하려면 소스 파일도 여러 개로 나누는 것이 필요
  - 헤더 파일도 프로그래머가 직접 만들 필요가 있음
    - 프로그래머가 직접 만든 헤더 파일을 사용자 정의 헤더파일
      - 사용자 정의 헤더파일도 시스템 헤더 파일과 같이 함수원형, 매크로, 자료형 재정의에 관련된 문장으로 구성

- **프로젝트 구성: chap16_linkedlist 폴더 생성**
  - 파일 linkedlist.c
    - 함수 main() 구현
  - 파일 listlib.c
    - 연결 리스트에 필요한 함수 createNode(), append(), printList() 구현
  - 헤더파일 linkedlist.h
    - 필요한 시스템 헤더 파일을 삽입
    - 함수 createNode(), append(), printList()의 함수 원형을 정의
    - 구조체 linked_list 정의와 관련된 자료 유형을 정의



프로그래밍 언어 종류를 연결 리스트로 구현

- **사용자 정의 헤더파일: linkedlist.h**
  - 표준 입출력에 필요한 헤더 파일 <stdio.h>추가
  - 두 구현 소스파일에서 헤더 파일 linkedlist.h을 추가
  - 지시자 #include에서 사용자 정의 헤더 파일을 삽입 방법: 큰 따옴표를 사용



linkedlist.h 파일

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// 자기참조 구조체 정의
struct linked_list
{
    char *name;
    struct linked_list *next;
};

typedef struct linked_list NODE;
typedef NODE *LINK;
```

listlib.c 파일

```c
#include "linkedlist.h"

LINK createNode(char *name)
{
   . . .
}

LINK append(LINK head, LINK cur)
{
   . . .
}

int printList(LINK head)
{
   . . .
}
```

linkedlist.c 파일

```c
#include "linkedlist.h"

void get_name(char *name, int len)
{
    . . .
}

int main(void)
{

    . . .

    return 0;
}
```

프로젝트 LinkedList 구현 내용: linkedlist.c

- **함수 main()**
  - 변수 char name[30]은 표준 입력으로 받은 프로그램 이름 문자열을 저장할 문자 배열
  - 변수 LINK head는 연결 리스트의 헤드로 사용
  - 변수 LINK cur는 현재 새로이 생성된 노드를 가리키는 포인터

- **함수 feof(stdin)**
  - 표준입력으로 문자열을 받고, 문장 while () 문을 이용
    - ctrl + Z를 누를 때까지 표준 입력을 계속 받음

- **함수 createNode()**
  - 문자열 name과 같은 문자열이 저장된 구조체 노드를 생성하여 변수 cur에 대입
  - 변수 cur를 인자로 함수 append()를 호출하여 연결 리스트에 추가
  - 새로운 노드가 하나 추가될 때마다 연결 리스트의 모든 노드를 순차적으로 출력

프로젝트 Linked List

- **연결 리스트를 구현한 프로그램**
  - 구현 소스: linkedlist.c, listlib.c

- **헤더파일: linkedlist.h 파일**

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct linked_list { //자기참조 구조체 정의
    char* name;
    struct linked_list* next;
};

typedef struct linked_list NODE; //struct linked_list를 NODE로 재정의
typedef NODE* LINK; //NODE *를 LINK로 재정의
```

listlib.c #1

```c
#include "linkedlist.h"

LINK createNode(char* name) //노드를 생성하는 함수
{
    //새로 생성되는 노드의 주소를 저장할 변수 cur를 선언
    LINK cur;
    cur = (LINK)malloc(sizeof(NODE));
    if (cur == NULL)
    {
        printf("노드 생성을 위한 메모리 할당에 문제가 있습니다.\n");
        return NULL;
    }

    //언어 이름을 저장할 문자배열을 동적 할당하여 name에 저장
    cur->name = (char*)malloc(sizeof(char) * (strlen(name) + 1));
    strcpy(cur->name, name);
    cur->next = NULL; //다음 노드는 모르므로 NULL로 저장

    return cur;
}
```

listlib.c #2

```c
//cur 노드를 연결 리스트 head의 마지막 노드에 추가하는 함수
LINK append(LINK head, LINK cur)
{
    //지역 변수 nextNode를 선언하고 초기 값으로 head를 저장
    LINK nextNode = head;

    //만일 현재 헤드가 가리키는 것이 없다면,
    // 즉 연결리스트의 노드가 하나도 없는 경우
    if (head == NULL)
    {
        head = cur; //추가하려는 노드가 head가 됨
        return head;
    }
    //멤버 next가 NULL일 때까지 이동하여 마지막 노드까지 이동
    while (nextNode->next != NULL)
    {
        nextNode = nextNode->next;
    }

    //추가 노드를 현재 노드의 next에 저장
    nextNode->next = cur;

    return head;
}
```

```c
//연결 리스트의 모든 노드 출력 함수
int printList(LINK head)
{
    int cnt = 0; //방문한 노드의 수를 저장
    LINK nextNode = head;

    //nextNode를 이용하여 연결리스트의 처음부터 끝까지 순회
    while (nextNode != NULL)
    {
        //리스트의 노드를 방문하여 방문 횟수와 문자열 자료를 출력
        printf("%3d번째 노드는 %s\n", ++cnt, nextNode->name);
        nextNode = nextNode->next;
    }

    return cnt; //총 노드 방문 횟수를 반환하고 함수를 종료
}
```

linkedlist.c

```c
#include "linkedlist.h"

extern LINK createNode(char* name);
extern LINK append(LINK head, LINK cur);
extern int printList(LINK head);
void get_name(char *name, int len)
{
    fgets(name, len, stdin);
    name[strlen(name) - 1] = '\0';
}

int main(void)
{
    char name[30]; //표준입력으로 받은 문자열을 저장할 문자 배열
    LINK head = NULL; //연결 리스트의 헤드로 사용
    LINK cur; //현재 새로이 생성된 노드를 가리키는 포인터

    printf("이름을 입력하고 Enter를 누르세요. >> \n");
    get_name(name, sizeof(name));
    while (!feof(stdin))
    {
        cur = createNode(name); //노드 동적 할당
        if (cur == NULL) {
            printf("동적메모리 할당에 문제가 있습니다.\n");
            exit(1);
        }
        head = append(head, cur); //맨 뒤에 노드 추가
        printList(head); //연결 리스트 모두 출력
        get_name(name, sizeof(name));
    }
    return 0;
}
```

listlib.c에 구현된 함수들
- 현재 파일에 없는 함수나 변수를 선언할 때 extern 키워드 사용

실행 파일 이름

컴파일 (Windows)

```
% gcc linkedlist.c listlib.c -o linkedlist.exe
```

컴파일 (Mac)

```
% gcc linkedlist.c listlib.c -o linkedlist
```

실행 결과

```
% linkedlist.exe 또는 ./linkedlist
이름을 입력하고 Enter를 누르세요. >>
C Enter
  1번째 노드는 C
C++ Enter
  1번째 노드는 C
  2번째 노드는 C++
Java Enter
  1번째 노드는 C
  2번째 노드는 C++
  3번째 노드는 Java
Python Enter
  1번째 노드는 C
  2번째 노드는 C++
  3번째 노드는 Java
  4번째 노드는 Python
Ctrl+Z 또는 Ctrl+D
```

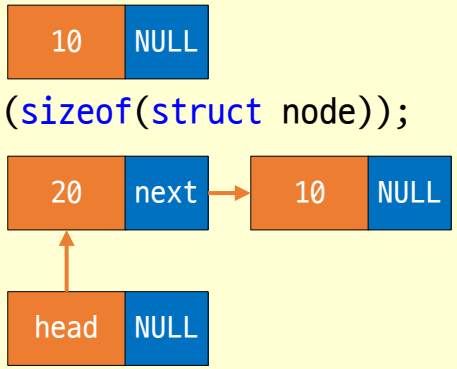
LAB 노드 3개로 구성된 연결 리스트(1)

- **구조체 struct node**
  - int형의 정보를 담는 x 필드
  - 다른 노드를 가리키는 next 필드로 구성
- **구현**
  - 먼저 노드 2개를 생성하여 연결 리스트로 연결
    - 포인터 head는 항상 연결 리스트의 첫 노드를 가리키게 하며
    - 마지막 노드의 필드 next는 NULL
  - 2개의 노드 중 마지막 노드로 이동하여 하나의 노드를 새로 생성
    - 필드 x를 30으로 저장하고 이 노드가 마지막 노드가 되도록
  - 연결 리스트의 모든 노드를 순회하면서 필드 x 값을 출력
- **결과**
  - 1번째 노드는 20
  - 2번째 노드는 10
  - 3번째 노드는 500

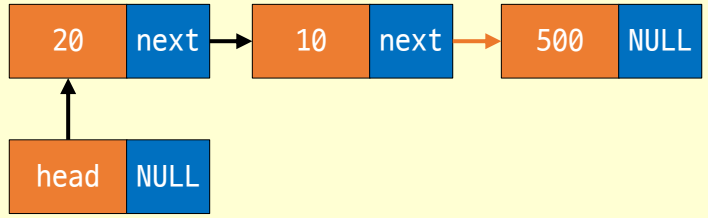
LAB 노드 3개로 구성된 연결 리스트(2)

<lab2nodelist.c>

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int x;
    struct node* next;
};

int main(void)
{
    //노드 두 개를 생성하여 자료와 링크를 대입
    struct node* one = malloc(sizeof(struct node));
    one->x = 10;
    one->next = NULL;
    struct node* two = malloc(sizeof(struct node));
    two->x = 20;
    two->next = one;

    struct node* head = two;
    struct node* cur = head;
    if (cur)
    {
        while (cur->next != NULL)
            cur = cur->next;
    }
```




```c
    //노드를 생성하여 주소를 저장
    cur->next = malloc(sizeof(struct node));
    cur = cur->next;

    cur->next = NULL;
    cur->x = 500;

    int cnt = 0;
    cur = head;
    while (cur != NULL)
    {
        //리스트의 노드를 방문하여 문자열 자료를 출력
        printf("%3d번째 노드는 %d\n", ++cnt, cur->x);
        cur = cur->next;
    }

    return 0;
}
```



실행 결과

```
1번째 노드는 20
2번째 노드는 10
3번째 노드는 500
```

Questions?