SCH 순천향대학교

자료구조 2

2018학년도 2학기

담당교수: 홍 민

수업 진행

- **목적:** 프로그램에 꼭 필요한 자료구조의 개념과 기본적인 알고리즘들을 학습하며 프로그램의 효율성을 분석하고 향상시킬 수 있는 방법을 배움
- **교재:** C 언어로 쉽게 풀어쓴 자료 구조 개정판, 천인국, 공용해, 하상호 저, 생능출판사, 2014
- **성적:** 중간고사 30%, 기말고사 30%, 과제물 30%, 퀴즈 10%
- **강의노트:** http://cyber.sch.ac.kr
- **강의 방법 및 report:** 강의 노트를 따라 진행. 과제는 매주에 한번 정도 프로그래밍과 문제 형식
- **절대 Cheating은 불가!!!**

수업 진행

- 7장 트리 부터, 우선순위 큐, 정렬, 그래프, 해싱, 탐색
- 자료구조 2 이론 수강생은 꼭 실습을 들어야 함
- 전체 수업의 1/3 이상 결석 시 무조건 F임
- 2번 지각하면 1번의 결석으로 처리
- 과제는 항상 자기 혼자의 힘으로 할 것
- 예습 및 복습을 철저히 할 것
- 항상 수업에 최선을 다하는 자세가 중요
- **절대 Cheating은 불가!!!**

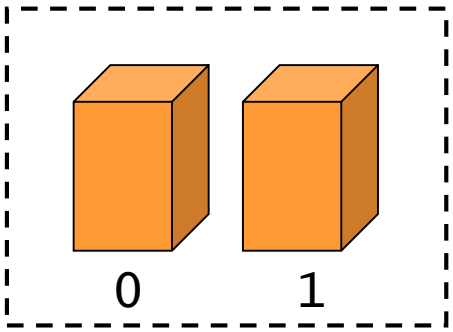
구조체 & 포인터

- 목차
  - 구조체
  - 포인터

구조체란?

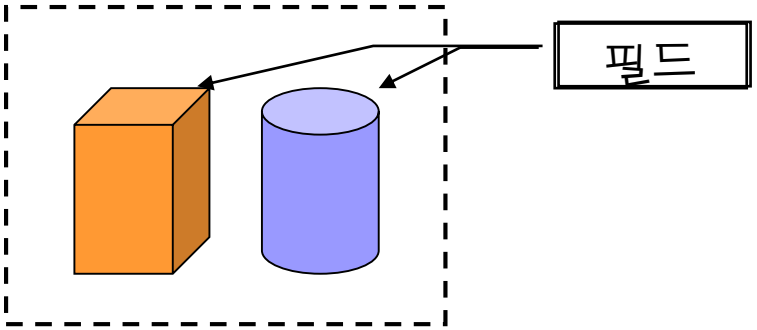
- 구조체(structure): 타입이 다른 데이터를 하나로 묶는 방법
- 배열(array): 타입이 같은 데이터들을 하나로 묶는 방법

배열




```
char carray[100];
```

구조체




```
struct example {
        char cfield;
        int ifield;
        float ffield;
        double dfield;
};
struct example s1;
```

구조체의 사용예

- 구조체의 선언과 구조체 변수의 생성

```
struct person {
        char name[10];      // 문자배열로 된 이름
        int age;            // 나이를 나타내는 정수값
        float height;       // 키를 나타내는 실수값
};
struct person a;            // 구조체 변수 선언
```

- typedef을 이용한 구조체의 선언과 구조체 변수의 생성

```
typedef struct person {
        char name[10];      // 문자배열로 된 이름
        int age;            // 나이를 나타내는 정수값
        float height;       // 키를 나타내는 실수값
} person;
person a;                   // 구조체 변수 선언
```

구조체의 대입과 비교 연산

- 구조체 변수의 대입: 가능

```
typedef struct person {
        char name[10];          // 문자배열로 된 이름
        int age;                // 나이를 나타내는 정수값
        float height;           // 키를 나타내는 실수값
} person;
main()
{
        person a, b;
        b = a;                  // 가능
}
```

- 구조체 변수끼리의 비교: 불가능

```
main()
{
        if( a > b )
                printf("a가 b보다 나이가 많음");    // 불가능
}
```

자체참조 구조체

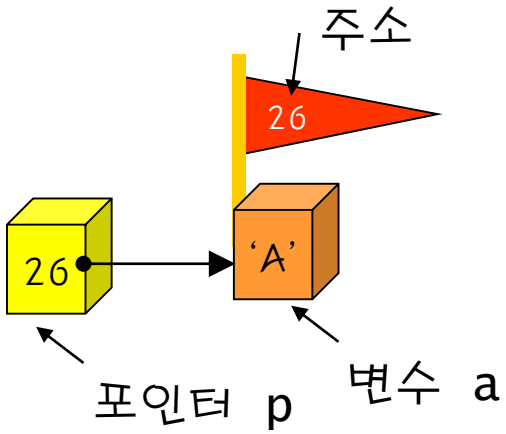
- 자체 참조 구조체(self-referential structure): 필드중에 자기 자신을 가리키는 포인터가 한 개 이상 존재하는 구조체
- 연결 리스트나 트리에 많이 등장

```
typedef struct ListNode {
        char        data[10];
        struct ListNode *link;
} ListNode;
```

포인터(Pointer)

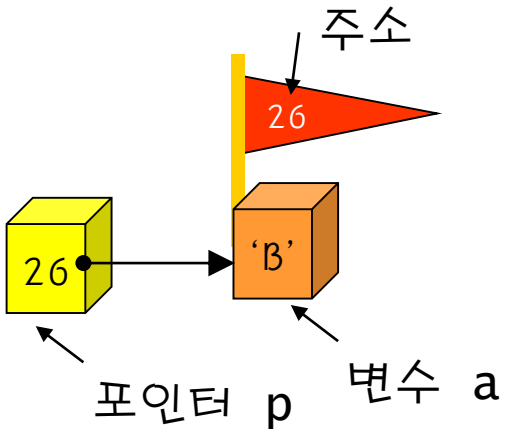
- 포인터: 다른 변수의 주소를 가지고 있는 변수

```
char a='A';
char *p;
p = &a;
```



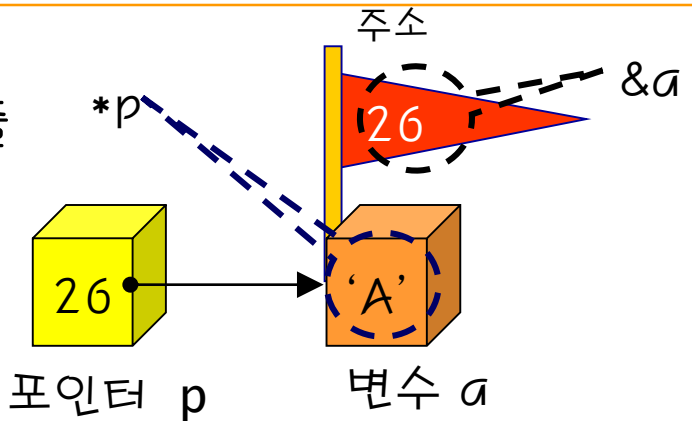
- 포인터가 가리키는 내용의 변경: * 연산자 사용

```
*p= 'B';
```



포인터와 관련된 연산자

- & 연산자: 변수의 주소를 추출
- * 연산자: 포인터가 가리키는 곳의 내용을 추출




```
 p       // 포인터
*p       // 포인터가 가리키는 값
*p++     // 포인터가 가리키는 값을 가져온 다음, 포인터를 한칸 증가한다.
*p--     // 포인터가 가리키는 값을 가져온 다음, 포인터를 한칸 감소한다.
(*p)++   // 포인터가 가리키는 값을 증가시킨다.
```

```
int a;   // 정수 변수 선언
int *p;  // 정수 포인터 선언
int **pp;          // 정수 포인터의 포인터 선언
p = &a;  // 변수 a와 포인터 p를 연결
pp = &p;           // 포인터 p와 포인터의 포인터 pp를 연결
```

다양한 포인터

- 포인터의 종류

```
void *p; // p는 아무것도 가리키지 않는 포인터
int *pi; // pi는 정수 변수를 가리키는 포인터
float *pf; // pf는 실수 변수를 가리키는 포인터
char *pc;  // pc는 문자 변수를 가리키는 포인터
int **pp;    // pp는 포인터를 가리키는 포인터
struct test *ps; // ps는 test 타입의 구조체를 가리키는 포인터
void (*f)(int) ; // f는 함수를 가리키는 포인터
```

- 포인터의 형변환: 필요할 때마다 형변환하는 것이 가능하다.

```
void *p;
pi=(int *) p;
```

문자열과 포인터

- 문자열 선언
  - 배열을 이용: char str1[9] = “Computer”;
  - 포인터 이용: char *str2 = “Computer”;

- 배열과 포인터 선언의 차이
  - str1과 str2는 모두 문자열의 처음인 ‘C’의 주소를 가리키고 있음
  - 배열로 선언된 str1은 다른 문자열을 가리킬 수 없으나 배열에 저장된 값을 다른 문자로 변경가능
  - 포인터로 선언된 str2는 다른 문자열을 가리킬 수 없으나 문자열의 내용은 변경이 안됨

문자열과 포인터 예제

```c
#include <stdio.h>  // 입출력을 위한 전처리명령문

void main()
{
    char str1[9] ="Computer";
    char *str2 ="Computer";

    printf("str1 = %s₩n", str1);
    printf("str2 = %s₩n₩n", str2);

    str2 = "Science";
    // str1 = "Science";  // 변경 불가능

    printf("str1 = %s₩n", str1);
    printf("str2 = %s₩n₩n", str2);

    str1[0] = 'c';
    // str2[0] = 'c';  // 변경 불가능

    printf("str1 = %s₩n", str1);
    printf("str2 = %s₩n₩n", str2);

}
```

함수의 파라미터로서의 포인터

- 함수안에서 파라미터로 전달된 포인터를 이용하여 외부 변수의 값 변경 가능

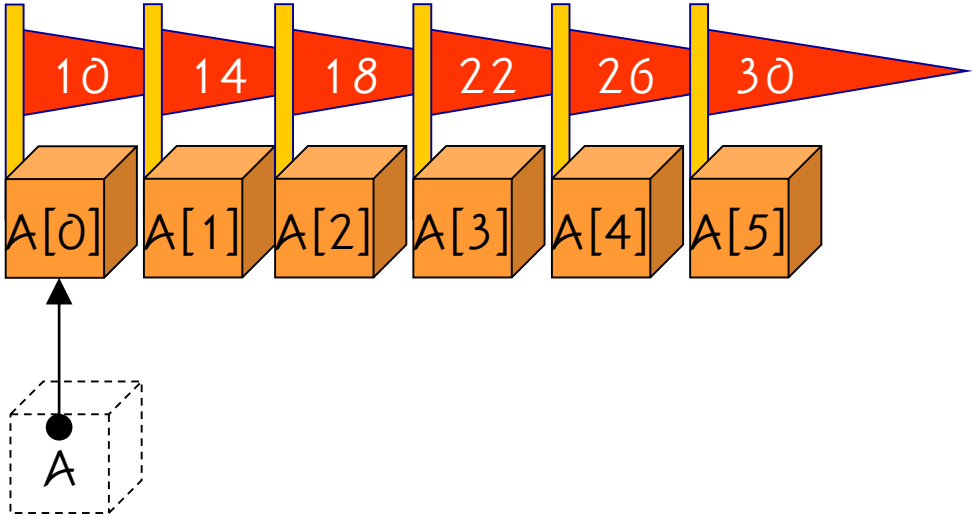
```
void swap(int *px, int *py)
{
        int tmp;
        tmp = *px;
        *px = *py;
        *py = tmp;
}
main()
{
        int a=1,b=2;
        printf("swap을 호출하기 전: a=%d, b=%d\n", a,b);
        swap(&a, &b);
        printf("swap을 호출한 다음: a=%d, b=%d\n", a,b);
}
```

포인터 사용 이유

- 변수의 지역성 해결
  - 변수는 생성된 함수내에서 사용 후 반납: stack 영역에 생성됨
  - 다른 함수에서 접근 불가
- 연속된 메모리 참조
  - 배열, 구조체와 같은 연속된 자료구조 참조
- 힢 영역 참조
  - 동적으로 할당된 힢(heap) 영역의 접근 가능

배열과 포인터

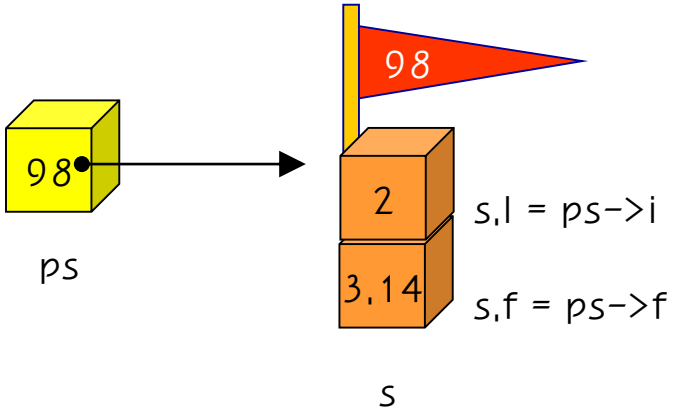
- 배열의 이름: 사실상의 포인터와 같은 역할



- 컴파일러가 배열의 이름을 배열의 첫번째 주소로 대치

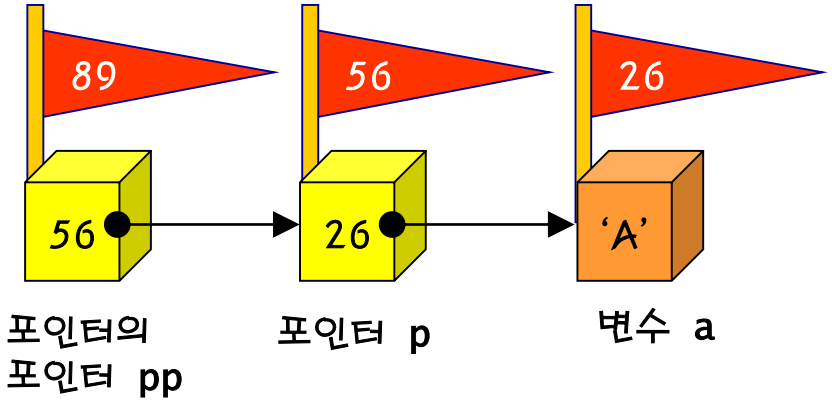
구조체의 포인터

- 구조체의 요소에 접근하는 포인터 연산자: ->




```
main()
{
        struct {
                int i;
                float f;
        } s, *ps;
        ps = &s;
        ps->i = 2;
        ps->f = 3.14;
}
```

포인터의 포인터

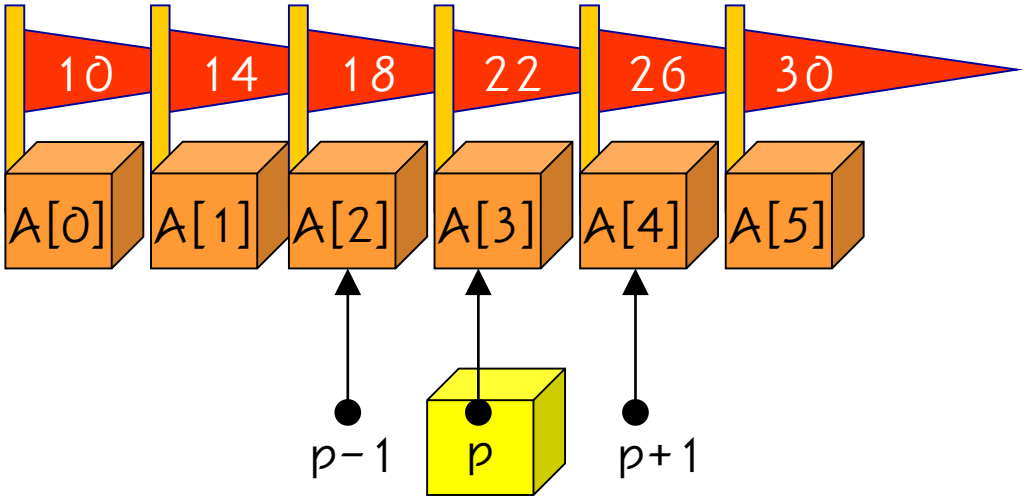



```
int a;   // 정수 변수 변수 선언
int *p;  // 정수 포인터 선언
int **pp;          // 정수 포인터의 포인터 선언
p = &a;  // 변수 a와 포인터 p를 연결
pp = &p;           // 포인터 p와 포인터의 포인터 pp를 연결
```

포인터 연산

- 포인터에 대한 사칙연산: 포인터가 가리키는 객체단위로 계산된다.

```
p       // 포인터
p+1     // 포인터 p가 가리키는 객체의 바로 뒤 객체
p-1     // 포인터 p가 가리키는 객체의 바로 앞 객체
```



포인터 사용시 주의할 점

- 포인터가 아무것도 가리키고 있지 않을 때는 NULL로 설정
  - int *pi = NULL;
- 초기화가 안된 상태에서 사용 금지

```
main()
{
        char *pc;    // 포인터 pi는 초기화가 안되어 있음
        *pc = 'E';   // 위험한 코드
}
```

- 포인터 타입간의 변환시에는 명시적인 타입 변환 사용

```
int *pi;
float *pf;
pf = (float *)pi;
```

포인터 예제

- 포인터 선언 및 활용에 대한 정확한 이해 필요
  - 다양한 사용 예제에 대한 이해

```c
#include <stdio.h>

void main()
{
    int a;
    int *p;
    int **pp;

    a = 100;
    p = &a;
    pp = &p;

    printf("a = %d\n", a);
    printf("&a = %d %x\n\n", &a, &a);


    printf("*p = %d\n", *p); // 포인터 p가 가리키고 있는곳에 저장된 값
    printf("p = %d %x\n", p, p); // 포인터 p에 저장된 주소값(a의 주소값)
    printf("&p = %d %x\n\n", &p, &p); // 포인터 p의 주소값

    printf("*pp = %d %x\n", *pp, *pp); // 포인터 pp가 가리키고 있는곳에 저장된 값
    printf("pp = %d %x\n", pp, pp); // 포인터 pp에 저장된 주소값(포인터 p의 주소값)
    printf("&pp = %d %x\n\n", &pp, &pp);  // 포인터 pp의 주소값

    printf("&*p = %d %x\n", &*p, &*p); // 포인터 p가 가리키고 있는곳의 주소(a의 주소값)
    printf("&*pp = %d %x\n", &*pp, &*pp); // 포인터 pp가 가리키고 있는곳의 주소(포인터 p의 주소값)
    printf("&**pp = %d %x\n\n", &**pp, &**pp); // a의 주소값

    // 변수 a에 저장된 값에 접근하는 방법
    printf("a = %d\n", a);
    printf("*p = %d\n", *p);
    printf("**pp = %d\n\n", **pp);

}
```

동적 메모리 할당

- 프로그램이 메모리를 할당받는 방법
  - 정적 메모리
  - 동적 메모리 할당
- 정적 메모리 할당
  - 메모리의 크기는 프로그램이 시작하기 전에 결정
  - 프로그램의 수행 도중에 그 크기가 변경될 수는 없다.
  - 만약 처음에 결정된 크기보다 더 큰 입력이 들어온다면 처리하지 못할 것이고 더 작은 입력이 들어온다면 남은 메모리 공간은 낭비될 것이다.
  - (예) 변수나 배열의 선언
  - int buffer[100];
char name[] = “data structure";
- 동적 메모리 할당
  - 프로그램의 실행 도중에 메모리를 할당받는 것
  - 필요한 만큼만 할당을 받고 또 필요한 때에 사용하고 반납
  - 메모리를 매우 효율적으로 사용가능

동적 메모리 할당

- 전형적인 동적 메모리 할당 코드

```
main()
{
    int *pi;
    pi = (int *)malloc(sizeof(int));   // 동적 메모리 할당
    …
    …                                  // 동적 메모리 사용
    …
    free(pi);                          // 동적 메모리 반납
}
```

- 동적 메모리 할당 관련 라이브러리 함수
  - malloc(size) // 메모리 할당
  - free(ptr) // 메모리 할당 해제
  - sizeof(var) // 변수나 타입의 크기 반환(바이트 단위)

동적 메모리 할당 라이브러리

- malloc(int size)
  - size 바이트 만큼의 메모리 블록을 할당

```
(char *)malloc(100) ;   /* 100 바이트로 100개의 문자를 저장 */
(int *)malloc(sizeof(int));/* 정수 1개를 저장할 메모리 확보*/
(struct Book *)malloc(sizeof(struct Book))/* 하나의 구조체 생성 */
```

- free(void ptr)
  - ptr이 가리키는 할당된 메모리 블록을 해제
- sizeof 키워드
  - 변수나 타입의 크기 반환(바이트 단위)

```
size_t  i = sizeof( int );                          // 4
struct AlignDepends {
    char c;
    int i;
};
size_t size = sizeof(struct AlignDepends);          // 8
int  array[] = { 1, 2, 3, 4, 5 };
size_t  sizearr = sizeof( array ) / sizeof( array[0] );   // 20/4=5
```

동적 메모리 할당 예제

```
struct Example {
        int number;
        char name[10];
};
void main()
{
        struct Example *p;

        p=(struct Example *)malloc(2*sizeof(struct Example));
        if(p==NULL){
          fprintf(stderr, "can't allocate memory\n") ;
          exit(1) ;
        }
        p->number=1;
        strcpy(p->name,"Park");
        (p+1)->number=2;
        strcpy((p+1)->name,"Kim");
        free(p);
}
```

동적 할당과 포인터

- 데이터 정렬을 위한 함수 사용
  - main 함수에서 필요한 만큼 동적으로 메모리를 할당
  - 정렬 실행 함수로 할당된 배열 사용

```
#include <stdio.h>  // 입출력을 위한 전처리명령문
#include <stdlib.h> // 동적할당을 위한 전처리명령문

void bubble_sort(int *list, int n)
{
    int i, j, temp;
    for(i = n-1; i > 0; i--)
    {
        for(j = 0; j < i; j++)
        {
            if(list[j] > list[j+1])
            {
                temp = list[j];
                list[j] = list[j+1];
                list[j+1] = temp;
            }
        }
    }
}

void main()
{
    FILE *fp;
    int *data;  // 파일안의 숫자의 자료형.
    int cnt=0;  // 숫자의 개수를 카운트 하기 위한 변수
    int i = 0,temp;
    int tot = 0;
    float avg;


    fp = fopen("data.txt" , "r");  // 파일개방
    if (fp == NULL)
    {
        printf("파일개방에러.");
        return ;
    }

    while(!feof(fp))    // 파일이 끝이 아니면 파일에서 데이터를 읽어옴
    {
```

이중 포인터

- 동적으로 이중배열을 구성하기 위해 사용
  - 필요한 만큼 동적으로 2중 배열 구조의 메모리 할당
- 포인터 변수값 변경
  - 포인터 변수가 가리키고 있는 주소값 변경에 사용

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


int main(void)
{

    FILE *fp;   // 파일포인터 변수선언
    int m_row = 0, m_col = 3;  // 행렬의 크기를 0으로 초기화
    int **matrix;
    int temp1, temp2, temp3;
    int i,j;

    fp = fopen("data.txt", "r");    // data.txt파일을 읽기모드로 열어 생성된 FILE구조체를 fp에 할당
    if(fp==NULL)    // 만약 fopen함수에서 에러가 발생하여 fp에 NULL값이 저장되었다면 파일열기실패
    {
        printf("파일에 열리지 않았습니다.₩n");
        return 0;   // 메세지 출력하고 프로그램 종료
    }

    // 행렬의 크기를 계산함
    while(!feof(fp))
    {
        fscanf(fp, "%d%d%d", &temp1, &temp2, &temp3);
        m_row++;
    }

    // Dynamic 배열의 크기에 따른 할당
    matrix = (int**) malloc(sizeof(int *) * m_row);
    for(i = 0; i < m_row; i++)
    {
        matrix[i] = (int*) malloc(sizeof(int ) * m_col);
    }
    rewind(fp);

    // 파일에서 읽어온 행렬의 크기를 가지고 그만큼의 행렬 내용을 읽어 옴
    for(i = 0; i < m_row; i++)  // 행을 증가 시킴
    {
```