11. 전처리문과 구조체

Autumn 2019

Today

1. 지난 시간 복습 – 함수와 포인터
2. 전처리문
3. 구조체
4. 실습

[복습] 포인터 변수

- 변수의 주소를 저장하는 변수

- 포인터 변수 생성
  - 데이터형에 * 을 붙여서 생성
  - int*, float*, double*, char*

- 다음 코드에 int 형 포인터 변수 p, q를 선언해서 각각 a, b의 주소를 저장한 뒤 다음의 scanf 와 printf 문을 바꿔보기

```c
#include <stdio.h>

int main()
{
    int a, b;

    puts("두 정수를 입력하세요.");
    scanf("%d %d", &a, &b);
    printf("두 수의 합은 %d입니다.\n", a + b);
}
```

```c
#include <stdio.h>

int main(){
    int a, b;
    int *p, *q;

    p = &a;
    q = &b;

    puts("두 정수를 입력하세요.");
    scanf("%d %d", p, q);
    printf("두 수의 합은 %d입니다. \n", *p + *q);
}
```

[복습] 문자열과 포인터

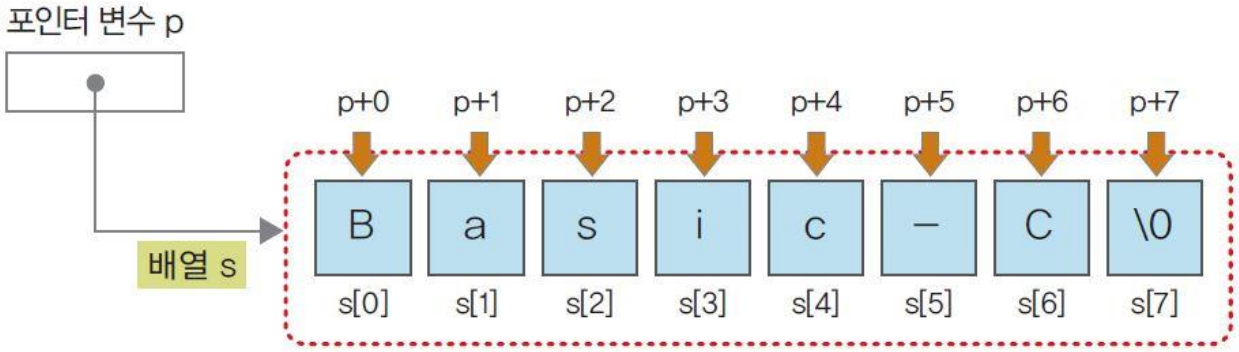
```c
int main(){
    char s[8] = "Basic-C";
    char* p;
    p = &s;
    printf("%s\n", p);

    printf("&s[3]: %s \n", &s[3]);
    printf("p+3: %s \n", p+3);

    printf("s[3]: %c \n", s[3]);
    printf("*(p+3): %c \n", *(p + 3));
}
```

p = s 와 동일함,

배열의 변수명은 주소!!




```
Basic-C
&s[3]: ic-C
p+3: ic-C
s[3]: i
*(p+3): i
```

그림 9-15 [응용 9-9]의 변수와 포인터의 관계

함수에 포인터 넘겨주기

- 두 경우의 차이는?
  - 매개변수로 값을 넘겨주는 경우 **vs.** 매개변수로 주소를 넘겨주는 경우
  - **(Call by value)** **(Call by reference)**

```c
void swap(int x, int y) {
    int temp = x;

    x = y;
    y = temp;
}

int main() {

    int a = 5, b = 7;

    swap(a, b);
    printf("a: %d, b:%d \n", a, b);
}
```

```c
void swap(int *x, int *y){
    int temp = *x;

    *x = *y;
    *y = temp;
}

int main(){

    int a = 5, b = 7;

    swap (&a, &b);
    printf("a: %d, b:%d \n", a, b);
}
```

Today

1. 지난 시간 복습 – 함수와 포인터
2. 전처리문
3. 구조체
4. 실습

전처리문 (preprocessor)

- 컴파일하기 전에 미리 처리되는 문장
- 소스코드 시작부분에 위치하며, #으로 시작
- 일반적으로 미리 정의되어 있음을 알아보기 쉽도록 대문자를 사용함
- 종류:
  #include, #define, #ifdef, #ifndef, #endif 등 다수

  예제. #include <stdio.h>

#define

- 소스코드에서 사용할 숫자, 문자열, 함수이름이 너무 길고 복잡할 때 상수처럼 사용할 기호를 정의해서 사용
- 사용법: #define [기호] [숫자, 문자열, 함수]
- 예시1.

```c
#include <stdio.h>

int main(){

    double r = 3.0;
    double area = r * r * 3.1415926535;

}
```

```c
#include <stdio.h>
#define PI 3.1415926535

int main(){

    double r = 3.0;
    double area = r * r * PI;

}
```

- 예시2.

```c
#include <stdio.h>
#define PI 3.1415926535
#define STR "원의 면적을 계산했습니다."
#define END_MSG printf("함수를 종료합니다.")

int main(){

    double r = 3.0;
    double area = r * r * PI;

    puts(STR);
    END_MSG;
}
```

숫자 전처리문

문자열 전처리문

함수 전처리문

#define

- 컴파일할 때 정의된 기호를 만나면 옆의 값으로 교체함

- 컴마가 없기 때문에 보통 작동을 안전하게 하기 위해 (괄호)를 사용함

```c
#include <stdio.h>
#define PI 3.1415926535
#define STR "원의 면적을 계산했습니다."
#define END_MSG printf("함수를 종료합니다.")

int main(){

    double r = 3.0;
    double area = r * r * PI;

    puts(STR);
    END_MSG;
}
```

```c
#include <stdio.h>
#define PI (3.1415926535)
#define STR ("원의 면적을 계산했습니다.")
#define END_MSG (printf("함수를 종료합니다."))

int main(){

    double r = 3.0;
    double area = r * r * PI;

    puts(STR);
    END_MSG;
}
```

#ifdef #ifndef #endif

- 문법의 조건문과 비슷하게 작동함
  - 이미 선언된 기호가 있는지 체크해서 컴파일 여부를 결정함
- 조건문 전처리기로, 컴파일 여부를 결정하도록 함

```c
#ifdef BLOCK
#include <stdio.h>
#define PI (3.1415926535)
#define STR ("원의 면적을 계산했습니다.")
#define END_MSG (printf("함수를 종료합니다."))

int main() {

    double r = 3.0;
    double area = r * r * PI;

    puts(STR);
    END_MSG;
}
#endif
```

```c
#define BLOCK
#ifdef BLOCK
#include <stdio.h>
#define PI (3.1415926535)
#define STR ("원의 면적을 계산했습니다.")
#define END_MSG (printf("함수를 종료합니다."))

int main() {

    double r = 3.0;
    double area = r * r * PI;

    puts(STR);
    END_MSG;
}
#endif
```

#ifdef #ifndef #endif

- 문법의 조건문과 비슷하게 작동함
  - 이미 선언된 기호가 있는지 체크해서 컴파일 여부를 결정함
- #ifdef와 반대로 작동함, 정의되어있지 않으면 컴파일하도록 함

```c
#define BLOCK
#ifndef BLOCK
#include <stdio.h>
#define PI (3.1415926535)
#define STR ("원의 면적을 계산했습니다.")
#define END_MSG (printf("함수를 종료합니다."))

int main() {

    double r = 3.0;
    double area = r * r * PI;

    puts(STR);
    END_MSG;
}
#endif
```

```c
#ifndef BLOCK
#define BLOCK
#include <stdio.h>
#define PI (3.1415926535)
#define STR ("원의 면적을 계산했습니다.")
#define END_MSG (printf("함수를 종료합니다."))

int main() {

    double r = 3.0;
    double area = r * r * PI;

    puts(STR);
    END_MSG;
}
#endif
```

Today

1. 지난 시간 복습 – 함수와 포인터
2. 전처리문
3. 구조체
4. 실습

구조체 (struct)

- 서로 다른 변수들을 하나로 묶어주는 기능

이름: 김xx	이름: 이xx	이름: 박xx	이름: 최xx	이름: char []
성별: 남	성별: 여	성별: 여	성별: 남	성별: char (M/F)
학번: 20170111	학번: 20131234	학번: 20151234	학번: 20161234	학번: int
학과: 산업공학	학과: 행정학과	학과: 경영학과	학과: 철학과	학과: int (0,1,..)
학점: 4.50	학점: 3.23	학점: 2.37	학점: 3.85	학점: double
..	..	..	..	..

....

- 혹은 비슷한 기능을 하는 변수끼리 묶어주고 싶을 때,

구조체 이름

```
struct student {
    char name[20];
    char gender;
    int number;
    double grade;
};
```

구조체에 포함되는 변수들 (member)

```
//날짜
struct date {
    int month;      //년
    int day;        //월
    int year;       //일
};
```

```
//좌표
struct point {
    int x;          //x좌표
    int y;          //y좌표
};
```

구조체 사용 예시

- 일반적으로 함수처럼 main 밖에서 선언하지만, main 안에서 구조체를 정의해도 무방함

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

struct student {
    char name[20];
    char gender;
    int number;
    double height;
};

int main()
{
    struct student s1;

    strcpy(s1.name, "Hwang");
    s1.gender = 'M';
    s1.number = 20170111;

}
```

구조체 형태 선언

구조체 변수 선언

데이터 대입

구조체 변수 선언하면서 초기화

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

struct student {
    char name[20];
    char gender;
    int number;
    double height;
};

int main() {

    struct student s1 =
    { "Hwang", 'F', 202021004, 178.4 };

    printf("%s: %c, %d, %lf",
        s1.name, s1.gender, s1.number, s1.height);
}
```

구조체 직접해보기

- 학생들의 프로그래밍1 과목의 각 항목별 점수를 저장하기 위해 구조체 Score를 정의해 보자. 정의된 구조체는 학번(int), 시험(50)(double), 과제(30)(double), 출석(20)(double) 점수를 포함해야 한다. 그리고 총점(100)(double) 및 학점(char)도 포함해야 한다.
- 다음으로 학생 1명에 대한 점수들을 키보드로 입력받아서 저장해 보자. 총점이 90점 이상이면 A, 80점 이상이면 B, 70점 이상이면 C, 70점 미만이면 D 학점을 저장하고, 학번과 학점을 출력해보자.

```
학번, 시험(50), 과제(30), 출석(20) 점수를 입력하세요.
20171234 42 28 18
20171234의 학점은 B입니다.
```

[실행 예]

구조체배열

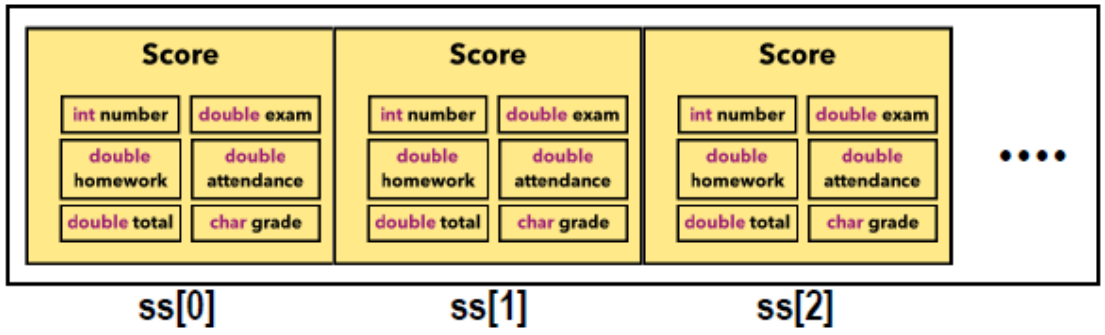
- 앞의 예제에서 학생 1명이 아니라 반 전체 40명의 점수 데이터를 저장하여 관리하려고 한다. 이때 40명에 대한 Score 구조체 변수를 각각 선언하는 것은 비효율적이다. 100명, 1000명이 된다면?

구조체를 배열로 선언할 수 있다

```
Score ss[40];
```

Score 구조체 40개를 배열로 선언

ss



구조체를 사용하는 이유는 기본적으로 많은 양의 데이터를 효율적으로 저장하고 관리하기 위해서이기 때문에, 구조체는 배열의 형태로 활용하는 것이 일반적임

구조체포인터

- 구조체포인터 변수는 구조체의 주소값을 저장한다.

❶ 일반 포인터 변수 사용

```
int a;
int* p;
p = &a;
*p = 100;
```

❷ 구조체 포인터 변수 사용

```
struct student {
    char name[10];
    int kor;
    int eng;
    float avg;
};
struct student s;
struct student *p;
p = &s;
p->kor = 100;
```

Today

1. 지난 시간 복습 – 함수와 포인터
2. 전처리문
3. 구조체
4. 실습

[실습1] 구조체

- 수강신청 프로그램에 교과목정보를 입력하는 구조체를 만들기
- 과목명은 길이 20, 교수이름은 길이 10으로 정의하시오.
- 교과목정보 구조체를 만들고, 아래 데이터 하나를 저장하고, 추가로 신청 가능한 학생수를 출력하시오.

과목명: Programming1

교수: Han

학년: 1

학점: 3

정원: 40

신청인원: 40

```
Programming1 과목의 여석은 0입니다.
```

[실습2] 구조체배열

- 앞의 실습에서 과목을 여러 개 받을 수 있도록 구조체 배열로 선언해보시오
- 다음의 정보를 키보드로 입력받고, 결과와 같이 출력하시오

과목명: Programming1	과목명: Calculus	과목명: Chemistry	과목명: English
교수: Han	교수: Kim	교수: Lee	교수: Park
학년: 1	학년: 1	학년: 1	학년: 1
학점: 3	학점: 3	학점: 3	학점: 2
정원: 40	정원: 55	정원: 27	정원: 20
신청인원: 40	신청인원: 45	신청인원: 25	신청인원: 17

```
1번째 교과목 정보 입력: 과목명, 교수, 학년, 학점, 정원, 신청인원
Programming1 Han 1 3 40 40
2번째 교과목 정보 입력: 과목명, 교수, 학년, 학점, 정원, 신청인원
Calculus Kim 1 3 55 45
3번째 교과목 정보 입력: 과목명, 교수, 학년, 학점, 정원, 신청인원
Chemistry Lee 1 3 27 25
4번째 교과목 정보 입력: 과목명, 교수, 학년, 학점, 정원, 신청인원
English Park 1 2 20 17
```

[실습3] 구조체함수

- 구조체를 매개변수로 사용해서 구조체의 정보를 출력하는 함수를 작성하시오

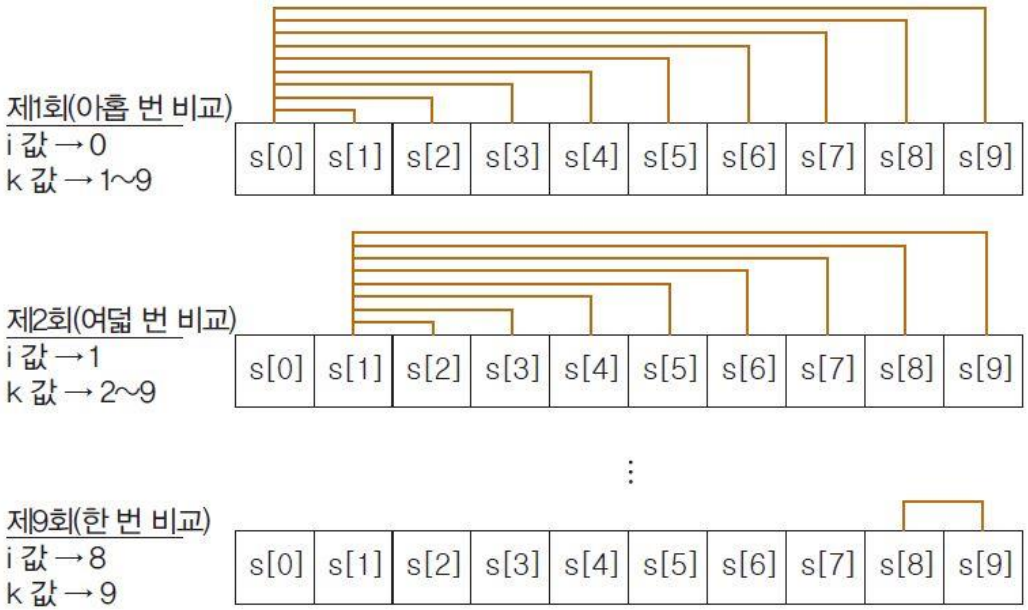
```
void printStatus(Course c)
{
    ..
}
```

- main() 에서 위의 함수를 호출해서 다음과 같이 결과를 출력해보시오 (printf 함수를 main 에서 사용하지 않습니다)

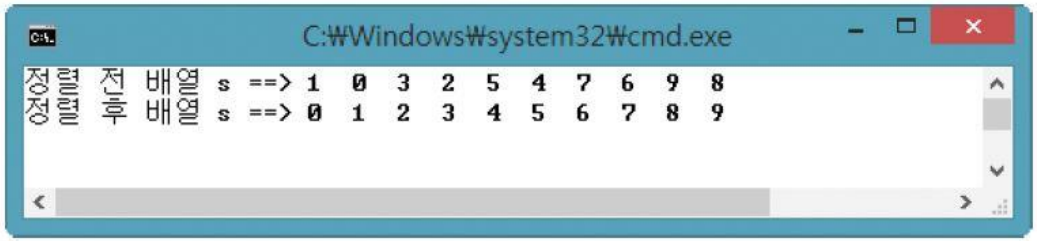
```
[수강신청 현황]
Programming1 - Han - 수강신청마감
Calculus - Kim - 10명 신청가능
Chemistry - Lee - 2명 신청가능
English - Park - 3명 신청가능
```

[실습26] 포인터를 이용한 배열 정렬

예제 설명 포인터를 이용하여 배열에 들어 있는 값 10개를 정렬하는 프로그램이다. 다음 그림을 참고하여 두 값을 비교하고 작은 것을 앞쪽으로 옮기는 선택 정렬을 사용한다.



실행 결과



[실습1]

- int형 배열에 0에서 999까지 random number를 10개 생성하여 저장한 후, 최소값을 계산하는 프로그램을 작성해 보세요.

```
0: 807
1: 249
2: 73
3: 658
4: 930
5: 272
6: 544
7: 878
8: 923
9: 709
최소값은  73입니다.
```

[실행 예]

[실습2]

- int형 배열에 0에서 999까지 random number를 100개 생성하여 저장한 후, 100단위별로 생성된 숫자들의 빈도수를 출력해 보세요.
- Hint. 앞의 예제를 확장하여 빈도수를 저장할 크기가 10인 int형 배열 bb를 생성하여 모두 0으로 초기화한다. 다음으로 for문을 사용하여 100개의 숫자들을 하나씩 반복하면서 각각 10개의 구간에 속하는 지를 체크하여, 속하는 구간의 빈도수를 1씩 증가시킨다.

```
0 - 99: 10개
100 - 199: 14개
200 - 299: 8개
300 - 399: 7개
400 - 499: 9개
500 - 599: 12개
600 - 699: 9개
700 - 799: 8개
800 - 899: 12개
900 - 999: 11개
```

[실행 예]