2020 C애플리케이션구현 포트폴리오

대학교	동양미래대학교
학과	컴퓨터정보공학과
학번	20190726
성명	한은진

목차

11. 문자와 문자열

[학습목표]

▶문자와 문자열을 이해하고 설명할 수 있다.
- 문자와 문자열의 표현과 저장 방법

▶문자와 문자열 입출력을 이해하고 설명할 수 있다.
- scanf(), printf(), getchar(), putchar(), getche(), getch(), putch()를 사용하여 문자 입출력
- scanf(), printf(), gets(), puts()를 사용하여 문자열 입출력

▶문자열 관련 함수를 이해하고 설명할 수 있다.
- 문자열 비교 함수 strcmp(), strncmp()를 사용하여 문자열 비교
- 문자열 연결 함수 strcat(), strncat()를 사용하여 문자열 연결
- 문자열 토큰 추출 함수 strtok()를 사용하여 문자열에서 토큰 추출
- 문자열 관련 함수 strlen(), strspn(), strcspn()의 사용 방법 이해
- 문자열 관현 함수 strlwr(), strupr()의 사용 방법 이해
- 문자열 관련 함수 strstr(), strchr()의 사용 방법 이해

▶여러 개의 문자열을 처리하는 방법에 대해 이해하고 설명할 수 있다.
- 문자 포인터 배열 방법과 이차원 문자 배열 방법의 차이
- 명령행 인자의 필요성과 구현 방법 이해

11. 문자와 문자열

- 문자와 문자열

개념	문자	영어의 알파벳이나 한글의 한 글자를 작은 따옴표와 같이 표기
	문자 상수	작은 따옴표에 의해 표기된 문자
	문자열	문자의 모임인 일련의 문자 / 큰따옴표로 표기

```c
(전역 범위)
#include <stdio.h>

int main(void)
{
    //문자 선언과 출력
    char ch = 'A';
    printf("%c₩n", ch); //문자 char형은 형식제어 문자 %c를 사용하여 출력

    //문자열 선언 방법1
    char java[] = { 'J','A','V','A','₩0' };
    char java2[5] = { 'J','A','V','A','₩0' };
    printf("%s₩n", java); //형식제어문자 %s를 사용하여 문자열 출력
    puts(java2);
    //문자열 선언 방법2
    char c[] = "C Language";
    printf("%s₩n", c);
    //문자열 선언 방법3
    char csharp[5] = "C#";
    printf("%s₩n", csharp);

    //문자 배열에서 문자 출력
    printf("%c%c₩n", csharp[0], csharp[1]);

    return 0;
}
```

권장

```
A
JAVA
JAVA
C Language
C#
C#
```

문자열의 마지막을 의미하는 NULL문자 '₩0'가 마지막에 저장되어야 한다.

배열크기는 반드시 저장될 문자 수보다 1이 커야 함->그래서 java2의 배열크기는 5

(같은 결과를 보여줌) -> puts()를 사용하면 한 줄에 출력한 후 다음 줄에서 출력을 준비

배열크기가 (문자수+1)보다 크면 나머지 부분은 모두 '₩0'문자로 채워진다.

11. 문자와 문자열

- 문자와 문자열

```c
#include <stdio.h>

int main(void)
{
    int i = 0;
    char* java = "java";
    printf("%s₩n", java);

    //수정 불가능, 실행오류 발생
    java[0] = 'J';

    return 0;
}
```

```
Microsoft Visual Studio 디버그 콘
java
```

문자 포인터에 의해 선언된 문자열은 문자 하나하나의 수정이 불가능하다 [중요 포인트]

문자열을 구성하는 문자 참조[포인터 사용] ↑

```c
#include <stdio.h>

int main(void)
{
    char c[] = "C C++ Java";          //문자열 선언
    printf("%s₩n", c);                //c의 값은 "C C++ Java"
    c[5] = '₩0';                      //문자 c[5]='₩0';에 의해 문자열이 2개로 나뉘어진다.
    printf("%s₩n%s₩n", c, (c + 6)); //c의 값은 "Java"가 된다.

    return 0;
}
```

```
Microsoft Visual Studio 디버그 콘솔
C C++ Java
C C++
Java
```

문자열은 문자열이 시작되는 부분부터 '₩0'문자까지

함수 printf()에서 %s는 문자 포인터가 가리키는 위치에서 NULL 문자까지를 하나의 문자열로 인식

'₩0'문자에 의한 문자열 분리 ↑

11. 문자와 문자열

- 문자와 문자열

다양한 문자 입력 함수

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

int main(void)
{
    char ch;

    printf("문자를 계속 입력하고 Enter를 누르면 >>Wn");
    while ((ch = getchar()) != 'q')
        putchar(ch);

    printf("Wn문자를 누를 때마다 두 번 출력 >>Wn");
    while ((ch = _getche()) != 'q')
        putchar(ch);

    printf("Wn문자를 누르면 한 번 출력 >>Wn");
    while ((ch = _getch()) != 'q')
        _putch(ch);
    printf("Wn");

    return 0;
}
```

getchar()함수를 사용하기 위해 <stdio.h>헤더파일 삽입

_getch()함수와 _getche()함수를 사용하기 위해 <conio.h>헤더파일 삽입

입력 반응

출력 반응

입출력 반응

출력 반응

```
C:₩Users₩user₩source₩repos₩Task₩Debug₩stringp
문자를 계속 입력하고 Enter를 누르면 >>
java
java
python
python
q

문자를 누를 때마다 두 번 출력 >>
jjaavvaaq
문자를 누르면 한 번 출력 >>
cprograming
```

Enter 후 반응 — 총 2번 보여짐

Enter 후 반응 — 총 2번 보여짐

버퍼를 사용하지 않고 문자 하나를 바로바로 입력할 수 있는 함수

입력한 문자가 화면에 보이지 않아 출력함수로 인해 출력

다양한 문자입력 함수	함수	scanf()	getchar()	getche() _getche()	getch() _getch()
	헤더파일	stdio.h		conio.h	
	버퍼 이용	버퍼 이용함		버퍼 이용 안함	
	반응	[enter]키를 눌러야 반응		문자마다 입력마다 반응	
	입력문자의 표시(echo)	누르면 바로 표시		누르면 바로 표시	표시 안됨
	입력문자 수정	가능		불가능	
	출력	printf()	putchar()		_putch()

Visual studio에서 _getche() _getch()함수를 사용

11. 문자와 문자열

- 문자와 문자열

gets()와 puts()

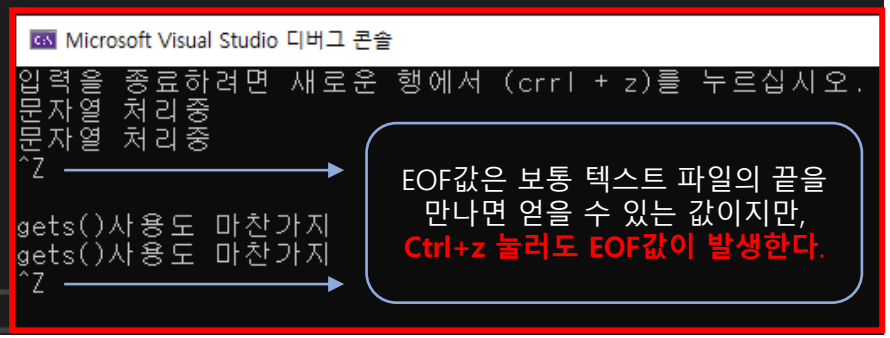
개념	**함수 gets()**	한 행의 문자열 입력에 유용한 함수 **Char * gets(char * buffer);**	**함수 gets_s()**	gets()의 대체함수로 사용을 권장 **Char * gets(char * buffer, size_t sizebuffer);**
	함수 puts()	한 행에 문자열을 출력하는 함수 **Int puts(const char * str);**		

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    char line[101]; //char *line 오류발생

    printf("입력을 종료하려면 새로운 행에서 (crrl + z)를 누르십시오. ₩n");
    while (gets(line))
        puts(line);
    printf("₩n");

    while (gets_s(line, 101))
        puts(line);
    printf("₩n");

    return 0;
}
```

문자열을 저장할 수 있는 충분한 공간의 문자 배열을 사용

gets()는 마지막에 입력된 '₩n'가 '₩0'로 교체되어 인자인 배열에 저장된다. 그러므로 프로그램에서 한 행을 하나의 문자열로 간주하고 프로그래밍 하도록!

puts()는 gets()와 반대로 문자열 마지막에 '₩0'을 '₩n'로 교체하여 버퍼에 전송해 모니터에 출력되면 한 행이 출력된다. 한 행 출력에 puts()가 효과적!
추가로, puts()는 오류를 발생하면 EOF를 반환



EOF값은 보통 텍스트 파일의 끝을 만나면 얻을 수 있는 값이지만,
Ctrl+z 눌러도 EOF값이 발생한다.

기호 상수 EOF(End Of File)는 파일의 끝이라는 의미로 stdio.h 헤더파일에 정수 -1로 정의되어 있다.

11. 문자와 문자열

- 문자와 문자열

문자열 입력 scanf()

```c
#define _CRT_SECURE_NO_WARNINGS //비주얼 스튜디오에서 안써주면 문제 발생
#include <stdio.h>
int main(void)
{
    //문자배열 변수
    char name[20], dept[30]; //char *name, *dept; 실행 오류 발생

    printf("%s", "학과 입력 >> ");
    scanf("%s", dept); //scanf -> enter키 반응
    printf("%s", "이름 입력 >> ");
    scanf("%s", name);
    printf("출력 : %10s %10s ₩n", dept, name);

    return 0;
}
```

문자열 입력은 충분한 공간의 문자배열이 있어야 가능
단순한 문자 포인터로는 문자열 저장이 불가능 **[중요 포인트]**

폭이 10이며, 우측 정렬로 된 문자열 출력

Microsoft Visual Studio 디버그 콘솔

```
학과 입력 >> 컴퓨터정보공학과
이름 입력 >> 한은진
출력 :  컴퓨터정보공학과       한은진
```

문자배열 변수로 scanf()에서 입력↑

**함수 printf()와 scanf()는 다양한 입출력에 적합하며,
문자열 입출력 함수 puts()와 gets()는 처리 속도가 빠르다는 장점이 있다.**

11. 문자와 문자열

- 문자열 관련 함수

다양한 문자열 라이브러리 함수

Strlen()	문자열의 길이를 반환하는 함수
Memcpy()	문자배열의 복사를 위한 함수
Memchr()	문자배열에서 문자 이후의 문자열을 찾는 함수
Memcmp()	문자배열에서 비교하는 함수

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

int main(void)
{
    char src[50] = "https://www.visualstudio.com"; //char형 배열 변수 선언 및 문자열 저장
    char dst[50];                       //char형 배열 변수 선언
    printf("문자배열 src = %sWn", src); //%s로 문자배열 src 출력
    printf("문자열크기 strlen(src) = %dWn", strlen(src)) ; //strlen함수로 인해 문자열 길이를 알 수 있음
    memcpy(dst, src, strlen(src) + 1);  //src에서 포인터 dst로 strlen(src)+1 수만큼 복사
                                        //strlne(src)+1에서 -1은 문자열의 마지막이 널이 되도록
    printf("문자배열 dst = %sWn", dst); //%s로 문자배열 dst 출력
    memcpy(src, "안녕하세요!", strlen("안녕하세요!") + 1); //안녕하세요! 문자열을 src로 복사
    printf("문자배열 src = %sWn", src); //출력

    char ch = ':';                      //구분자의 문자 : 를 변수 ch에 저장
    char* ret;                          //문자 포인터 변수 ret 선언
    ret = memchr(dst, ch, strlen(dst)); //memchr 로출로 문자열 dst문자 : 이후의 문자열을 반환하여 ret에 저장
    printf("문자 %c 뒤에는 문자열 %s 이 있다.Wn", ch, ret); //출력

    return 0;
}
```

```
선택 Microsoft Visual Studio 디버그 콘솔
문자배열 src = https://www.visualstudio.com
문자열크기 strlen(src) = 28
문자배열 dst = https://www.visualstudio.com
문자배열 src = 안녕하세요!
문자 : 뒤에는 문자열 ://www.visualstudio.com 이 있다.
```

11. 문자와 문자열

- 문자열 관련 함수

문자열 비교와 복사, 복사, 그리고 문자열 연결 등과 같은 다양한 문자열 처리는 헤더파일 string.h에 함수원형으로 선언된 라이브러리 함수로 제공

두 문자열 비교하는 함수 : strcmp()

Int strcmp(const char *s1, const char *s2);	두 인자인 문자열에서 같은 위치의 문자를 앞에서부터 다를 때까지 비교하여 **같으면 0을 반환하고, 앞이 크면 양수를, 뒤가 크면 음수를 반환**
Int strcnmp(const char *s1, const char *s2, size_t maxn);	두 인자 문자열을 같은 위치의 문자를 앞에서부터 다를 때까지 비교하나 **최대 n까지만** 비교하여 **같으면 0을 반환, 앞이 크면 양수를, 뒤가 크면 음수를 반환**

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

int main(void) {
    char* s1 = "java";
    char* s2 = "java";
    printf("strcmp(%s, %s) = %d₩n", s1, s2, strcmp(s1, s2));

    s1 = "java";
    s2 = "jav";
    printf("strcmp(%s %s) = %d₩n", s1, s2, strcmp(s1, s2));
    s1 = "jav";
    s2 = "java";
    printf("strcmp(%s %s) = %d₩n", s1, s2, strcmp(s1, s2));
    printf("strncmp(%s,%s,%d) = %d₩n", s1, s2, 3, strncmp(s1, s2, 3));

    return 0;
}
```

값이 같아 0

앞이 커 양수

뒤가 커 음수

3번째까지 같아 0

```
Microsoft Visual Studio 디버그 콘솔
strcmp(java, java) = 0
strcmp(java jav) = 1
strcmp(jav java) = -1
strncmp(jav,java,3) = 0
```

11. 문자와 문자열

- 문자열 관련 함수

문자열을 복사는 함수 : strcpy()

Char *strcpy(char *dest, const char *source);	• 앞 문자열 dest에 처음에 뒤 문자열 null문자를 포함한 source를 복사하여 그 복사된 문자열을 반환 • 앞 문자열은 수정되지만 뒤 문자열은 수정될 수 없다.
Char *strncpy(char *dest, const char *source, size_t maxn);	• 앞 문자열 dest에 처음에 뒤 문자열 source에서 **n개 문자를 복사하여 그 복사된 문자열을 반환** • 만일 지정된 maxn이 source의 길이보다 길면 나머지는 모두 널 문자가 복사된다. **앞 문자열은 수정되지만 뒤 문자열은 수정될 수 없다.**
Errno_t strcpy_s(char *dest, size_t sizedest, const char *source); **Errno_t strncpy_s(char *dest, size_t sizedest, const char *source, size_t maxn);**	• 두 번째 인자인 sizedest는 정수형으로 dest의 **크기를 입력**한다. • 반환형 errno_t는 정수형이며 반환값은 오류번호로 성공하며 0을 반환 • 비주얼 스튜디오에서 이 함수를 사용하길 권장

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

int main(void) {
    char dest[80] = "Java";
    char source[80] = "C is a language.";

    printf("%s₩n", strcpy(dest, source));
    printf("%s₩n", strncpy(dest, "C#", 2));
    printf("%s₩n", strncpy(dest, "C#", 3));

    return 0;
}
```

충분한 공간을 확보

Source를 dest에 모두 복사

C#을 dest에 2바이트만 복사

C#을 dest에 3바이트까지 복사

Microsoft Visual Studio 디버그 콘솔

```
C is a language.
C#is a language.
C#
```

문자열 관련 함수에서 단순히 문자열 포인터를 수정이 가능한 문자열의 인자로 사용할 수 없다.
즉, strcpy()와 strcat()에서 첫 인자로 문자열 포인터변수는 사용 불가능

11. 문자와 문자열

- 문자열 관련 함수

문자열 뒤에 다른 하나의 문자열을 추가해 연결하는 함수 : strcat()

Char *strcat(char *dest, const char *source);	• 앞 문자열 dest에 뒤 문자열 source를 연결해 저장하며, **이 연결된 문자열을 반환하고 뒤 문자열을 수정될 수 없다.**
Char *strncat(char *dest, const char *source, size_t maxn);	• 앞 문자열 dest에 뒤 문자열 source중에서 **n개의 크기 만큼을 연결해 저장**하며, 이 연결된 문자열을 반환하고 뒤 문자열은 수정될 수 없다. • 지정한 maxn이 문자열 길이보다 크면 null 문자까지 연결한다.
Errno_t strcat_s(char *dest, size_t sizedest, const char *source); **Errno_t strncat_s(char *dest, size_t sizedest, const char *source, size_t maxn);**	• 두 번째 인자인 sizedest는 정수형으로 dest의 크기를 입력한다. • 반환형 errno_t는 정수형이며 반환값은 오류번호로 성공하면 0을 반환 • 비주얼 스튜디오에서는 앞으로 이 함수의 사용을 권장한다.

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

int main(void) {
    char dest[80] = "C";

    printf("%s\n", strcat(dest, " is "));
    printf("%s\n", strncat(dest, "a java", 2));
    printf("%s\n", strcat(dest, "procedural "));
    printf("%s\n", strcat(dest, "language."));

    return 0;
}
```

충분한 공간을 확보

dest 뒤 ' is '연결

dest 뒤 a java에서 2개만 연결

dest 뒤 연결

dest 뒤 연결

Microsoft Visual Studio 디버그 콘솔

```
C is
C is a
C is a procedural
C is a procedural language.
```

문자열 관련 함수에서 단순히 문자열 포인터를 수정이 가능한 문자열의 인자로 사용할 수 없다.
즉, strcpy()와 strcat()에서 첫 인자로 문자열 포인터변수는 사용 불가능

11. 문자와 문자열

- 문자열 관련 함수

문자열에서 구분자인 문자를 여러 개 지정하여 토큰을 추출하는 함수 : strtok()

Char *strtok(char *str, const char *delim);	• 앞 문자열 str에서 뒤 문자열 delim을 구성하는 구분자를 기준으로 순서대로 토큰을 추출하여 반환하는 함수이며, 뒤 문자열 delim은 수정될 수 없다.
Char *strtok_s(char *str, const char *delim, char **context);	• 마지막 인자인 context는 함수 호출에 사용되는 위치 정보를 위한 인자이며, 비주얼 스튜디오에서도 이 함수를 사용을 권장한다.

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

int main(void) {
    char str1[] = "C and C++₩t language are best!";
    char* delimiter = " ,₩t!";

    printf("문자열 ₩" %s₩"을 >>₩n", str1);
    printf("구분자[%s]를 이용하여 토큰을 추출 >> ₩n", delimiter);
    char* ptoken = strtok(str1, delimiter);
    while (ptoken != NULL) {
        printf("%s₩n", ptoken);
        ptoken = strtok(NULL, delimiter);
    }

    return 0;
}
```

구분자 공백,쉼표,탭,느낌표 총4개 이다.

문자열 str1을 구분자를 기준으로 추출

널 값이 아닐 때까지 반복

```
Microsoft Visual Studio 디버그 콘솔
문자열 " C and C++      language are best!"을 >>
구분자[ ,       !]를 이용하여 토큰을 추출 >>
C
and
C++
language
are
best
```

그 외 다양한 문자열 관련 함수

strlwr()	문자열 모두 소문자로 변환
strupr()	문자열 모두 대분자로 변환

11. 문자와 문자열

- 여러 문자열 처리

문자 포인터 배열

하나의 문자 포인터가 하나의 문자열을 참조할 수 있으므로 **문자 포인터 배열은 여러 개의 문자열을 참조할 수 있다.**
장점 : 문자열 저장을 위한 최적의 공간을 사용한다는 점
단점 : 문자 포인터를 사용해서는 문자열 상수의 수정이 불가능하다는 점

```
char* pa[] = { "JAVA", "C#", "C++" };
//각각의 문자열 출력
printf("%s ", pa[0]); printf("%s ", pa[1]); printf("%s ", pa[2]);
```

이차원 문자 배열

배열 선언에서 이차원 배열의 열 크기는 문자열 중에서 가장 긴 문자열의 길이보다 1 크게 지정해야 한다.
장점 : 문자열을 수정할 수 있다는 점
단점 : 모든 열 수가 동일하게 할당되기 때문에 열의 길이 다른 경우 '₩0'값이 들어가 메모리 공간이 낭비된다는 점

```
char ca[][5] = { "JAVA", "C#", "C++" };
//각각의 3개 문자열 출력
printf("%s ", ca[0]); printf("%s ", ca[1]); printf("%s ", ca[2]);
```

첫 번째(행) 크기는 문자열 개수를 지정하거나 빈 공백으로 두며,
두 번째(열) 크기는 문자열 중에서 가장 긴 문자열의 길이보다 1크게 지정한다.
반드시, 열의 크기는 지정해줘야 오류가 뜨지 않는다.

12. 변수 유효범위

[학습목표]

▶변수 유효범위를 이해하고 설명할 수 있다.

- 지역변수, 자동변수를 선언하고 사용
- 전역변수를 선언하고 사용
- 키워드 extern의 필요성과 사용 방법

▶정적 변수와 레지스터 변수를 이해하고 설명할 수 있다.

- 기억 부류 auto, static, register, extern
- 지역 정적 변수와 전역 정적 변수의 필요성과 사용

12. 변수 유효범위

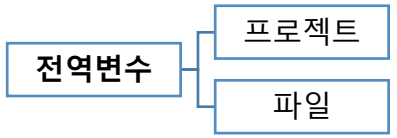
- 전역변수와 지역변수

변수 scope(범위)

변수의 참조가 유효한 범위를 변수의 유효 범위라고 한다.
변수의 유효 범위는 크게 **지역 유효 범위**(local scope)와 **전역 유효 범위**(global scope)로 나눌 수 있다.

지역 유효 범위 :
함수 또는 블록 내부에서 선언되어 그 지역에서 변수의 참조가 가능한 범위

전역 유효 범위 :
하나의 파일에서만 변수의 참조가 가능한 범위와 프로젝트를 구성하는 모든 파일에서 변수의 참조가 가능한 범위



↑ 변수의 유효범위

지역 변수

개념 : 지역변수는 함수 또는 블록에서 선언된 변수이다. 내부변수 또는 자동변수로도 부른다.

특징

- **함수나 블록에서 지역변수는 선언 문장 이후에 함수나 블록의 내부에서만 사용이 가능하다.**
- **함수의 매개변수도 함수 전체에서 사용 가능한 지역변수와 같다.**
- **지역변수는 선언 후 초기화하지 않으면 쓰레기 값이 저장되므로 주의해야 한다.**
- 지역변수는 그 변수가 선언된 함수 또는 블록에서 선언 문장이 실행되는 시점에서 메모리에 할당

스택 : 지역변수가 할당되는 메모리 영역
지역변수는 선언된 부분에서 자동으로 생성되고
종료되는 순간 메모리에서 자동으로 제거된다.
지역변수 선언에서 자료형 앞에 키워드 auto가 사용될 수 있다.
auto 생략 가능하여 일반적으로 없는 경우가 대부분

12. 변수 유효범위

- 전역변수와 지역변수

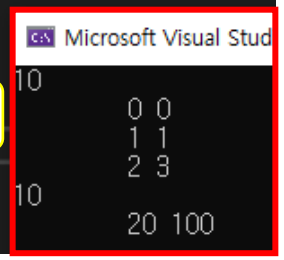
```c
#include <stdio.h>

void sub(int param); //함수

int main(void) {
    auto int n = 10; //지역변수 선언
    printf("%d₩n", n); //10출력

    //m, sum은 for문 내부의 블록 지역변수
    for (int m = 0, sum = 0; m < 3; m++) //3번 반복
    {
        sum += m; //sum 나온 숫자의 누적의 합
        printf("₩t%d %d₩n", m, sum); //tab + sum출력
    }

    printf("%d₩n", n); //n 참조 가능

    //함수 호출
    sub(20);

    return 0;
}

//매개변수인 param도 지역 변수와 같이 사용
void sub(int param)
{
    //지역변수 local
    auto int local = 100;
    //param과 local 참조 가능
    printf("₩t%d %d₩n", param, local);
}
```

지역변수 n의 영역 유효 범위

지역변수 sum, m 의 영역 유효 범위

지역변수 param 영역 유효 범위

지역변수 local 영역 유효 범위



12. 변수 유효범위

- 전역변수와 지역변수

전역변수

개념 : 전역변수는 함수 외부에서 선언되는 변수이다 외부변수라고도 부른다.

특징 :

- **일반적으로 파일 내부 또는 프로젝트 전체에서 참조할 수 있다.**
- 전역변수는 선언되면 자동으로 초기값이 자료형에 맞는 0으로 지정된다.
- 함수나 블록에서 전역변수와 같은 이름으로 지역변수를 선언 가능하다. 이런 경우, 함수 내부나 블록에서는 지역함수로 인식
- 전역변수는 프로젝트의 다른 파일에서도 참조 가능하다. 이런 경우, 키워드 extern을 사용하여 선언해야 한다.

extern을 사용한 참조선언 구문은 변수선언 문장 앞에 extern을 넣는 구조다.
extern 참조선언 구문에서 자료형은 생략할 수 있으나 쓰는 것을 권장
extern을 사용한 변수 선언은 이미 존재하는 전역변수의 유효 범위를 확장하는 것이다.

장점 : 전역변수는 어디서든지 수정할 수 있으므로 사용이 편하다.
단점 : 전역변수에 예상하지 못한 값이 저장된다면 프로그램 어느 부분에서 수정되었는지 알기 어렵다.

→ 이러한 문제로 전역변수는 가능한 제한적으로 사용하는 것을 권장한다.

12. 변수 유효범위

- 전역변수와 지역변수

```c
#include <stdio.h>

double getArea(double); //함수원형
double getCircum(double);

//전역변수 선언
double PI = 3.14;
int gi;

int main(void) {
    //지역변수 선언
    double r = 5.87;
    //전역변수 PI와 같은 이름의 지역변수 선언
    const double PI = 3.141592;

    printf("면적: %.2f₩n", getArea(r));
    printf("둘레1: %.2f₩n", 2 * PI * r);
    printf("둘레2: %.2f₩n", getCircum(r));
    printf("PI: %f₩n", PI); //지역변수 PI 참조
    printf("gi: %d₩n", gi); //전역변수 gi 기본값

    return 0;
}

double getArea(double r) //함수정의
{
    return r * r * PI; //전역변수 PI 참조
}
```

전역변수 PI 유효 범위

지역변수 PI 유효 범위

외부 프로젝트

```c
//이미 외부에서 선언된 전역변수임을 알리는 선언
extern double PI  //double은 생략은 가능하나 써주는 것을 권장

double getCircum(double r)
{
    return 2 * r * PI; //전역변수 PI 참조
}
```

전역변수 PI 유효 범위

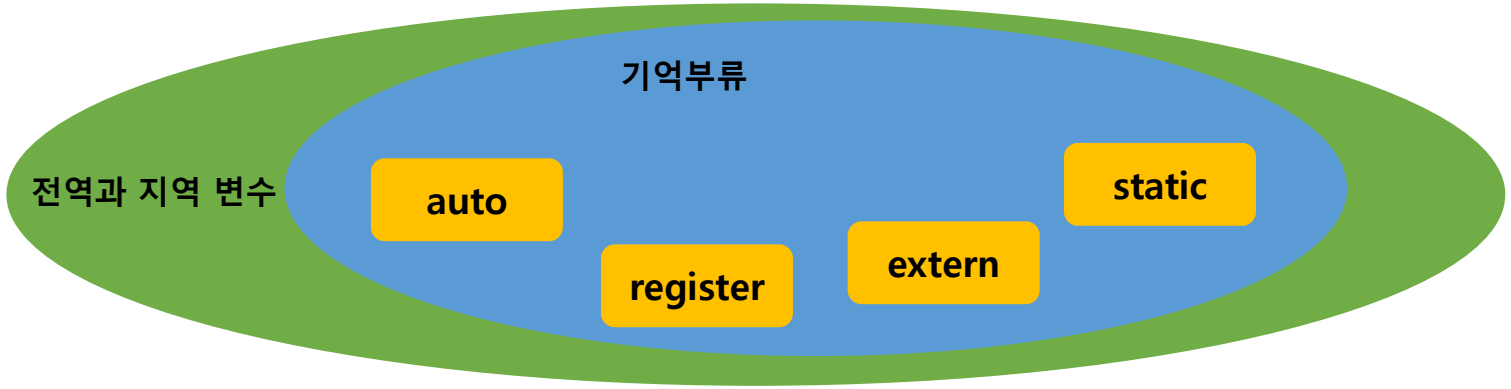
다른 파일에서 선언된
전역변수임을 알리는 변수 선언

12. 변수 유효범위

- 정적 변수와 레지스터 변수

기억부류와 레지스터 변수

변수는 4가지의 기억부류인 auto, register, static, extern에 따라 할당되는 메모리 영역이 결정되고 메모리의 할당과 제거 시기가 결정된다.
모든 기억 부류는 전역 변수 또는 지역 변수이다.



[기억부류 종류와 유효 범위↓]

기억부류 종류	전역	지역
auto	X	O
register	X	O
static	O	O
extern	O	x

키워드 extern을 제외하고
나머지는 변수선언에서 초기값을 저장 가능

12. 변수 유효범위

- 정적 변수와 레지스터 변수

키워드 register

개념 : 레지스터 변수는 변수의 저장공간이 일반 메모리가 아니라 CPU 내부의 레지스터에 할당되는 변수이다.

특징

- 키워드 register를 자료형 앞에 넣어 선언한다.
- **레지스터 변수는 지역변수로서 함수나 블록이 시작되면 CPU 내부 레지스터 값이 저장되고, 함수나 블록을 빠져나오면 소멸된다.**
- **처리 속도가 빠른 장점이 있다. -> 반복문 횟수를 제어하는 제어변수에 효과적**
- 레지스터 변수는 일반 메모리에 할당되는 변수가 아니므로 주소연산자 &를 사용할 수 없다.
- 시스템의 레지스터는 그 수가 한정되어 있어 레지스터 변수로 선언하더라도 레지스터가 모자라면 일반 지역변수로 할당된다.
- 레지스터 변수는 일반 지역변수와 같이 초기값을 저장되지 않으면 쓰레기값이 저장된다.

정적 변수

키워드 static

변수 선언에서 static을 넣어 정적변수를 선언
정적변수는 **정적 지역변수**와 **정적 전역변수**로 나눌 수 있다.

특징

- 정적변수는 초기 생성 이후 메모리에서 제거되지 않으므로 지속적으로 저장 값을 유지하거나 수정할 수 있는 특징이 있다.
- 프로그램이 시작되면 메모리에 할당되고, 프로그램이 종료되면 메모리에서 제거된다.
- **정적변수는 초기값을 지정하지 않으면 자동으로 자료형에 따라 0으로 null값이 저장된다. ->전역변수와 공통점**
- **정적변수의 초기화는 단 한번만 수행된다. 한번 초기화된 정적변수는 실행 중간에 더 이상 초기화하지 않는 특성이 있다. ->전역변수와 차이점**
- **주의할 점은 초기화는 상수로만 가능하다는 것이다.**

[초기화 주의]
정적 변수의 초기값에 변수를 대입하게 되면 초기화 문법 오류가 발생
그러므로 상수로만 대입

12. 변수 유효범위

- 정적 변수와 레지스터 변수

정적 지역변수

함수나 블록에서 정적으로 선언되는 변수가 정적 지역변수이다.
정적 지역변수는 함수나 블록을 종료해도 메모리에서 제거되지 않고 계속 메모리에 유지 관리되는 특성이 있다.

```c
#include <stdio.h>

void increment(void);//함수원형

int main(void) {
    //자동 지역변수
    for (int count = 0; count < 3; count++)
        increment(); //3번 함수호출
}

void increment(void) {
    static int sindex = 1;  //정적 지역변수
    auto int aindex = 1;    //자동 지역변수

    printf("정적 지역변수 sindex: %2d,₩t", sindex++);
    printf("자동 지역변수 aindex: %2d₩n", aindex++);
}
```

정적변수의 차이를 보여주는 코드

```
Microsoft Visual Studio 디버그 콘솔
정적 지역변수 sindex:  1,        자동 지역변수 aindex:  1
정적 지역변수 sindex:  2,        자동 지역변수 aindex:  1
정적 지역변수 sindex:  3,        자동 지역변수 aindex:  1
```

12. 변수 유효범위

- 정적 변수와 레지스터 변수

정적 전역변수

함수 외부에서 정적으로 선언되는 변수가 정적 전역 변수이다.
일반 전역변수는 다른 파일에서 extern을 사용하여 참조가 가능하다. 그러나 **정적 전역변수는 선언된 파일 내부에서만 참조가 가능한 변수이다. 즉, 정적 전역변수는 extern에 의해 다른 파일에서 참조가 불가능하다.**

```c
#include <stdio.h>

static int svar; //정적 전역변수 선언
int gvar; //전역변수 선언

void increment(); //함수 원형
void testglobal(); //함수 원형

int main(void) {
    for (int count = 1; count <= 5; count++)
        increment();
    printf("함수 increment()가 총 %d번 호출되었습니다.₩n", svar);

    testglobal(); //외부파일 함수
    printf("전역 변수 : %d₩n", gvar);
}

//함수 구현
void increment(){
    svar++; //수 증가
}
```

```c
void teststatic() {
    //정적 전역변수는 선언 및 사용 불가능
    //extern svar;
    //svar = 5;
}

void testglobal() {
    //전역변수는 선언 및 사용 가능
    extern gvar;
    gvar = 10;
}
```

정적 전역변수는
외부 파일에서 참조가
불가능하기 때문에 주석처리

프로그램이 크고 복잡하면 전역변수의 사용은 원하지 않는 전역변수의 수정과 같은 부작용의 위험성이 항상 존재한다. 그러므로 가급적 사용을 자제하는 것이 좋다.

12. 변수 유효범위

- 메모리 영역

메모리 영역

메인 메모리의 영역은 프로그램 실행 과정에서 데이터 영역, 힙 영역, 스택 영역 세 부분으로 나뉜다.
메모리 영역은 변수의 유효범위와 생존기간에 결정적 역할을 하며, 변수의 기억부류에 따라 할당되는 메모리 공간이 달라진다.

- **메모리 영역**
  - **데이터 영역** — 전역변수와 정적변수가 할당되는 저장공간
  - **힙 영역** — 동적 할당되는 변수가 할당되는 저장공간
  - **스택 영역** — 함수 호출에 의한 형식 매개변수 그리고 함수 내부의 지역변수가 할당되는 저장공간

스택 영역은 메모리 주소가 높은 값에서 낮은 값으로 저장 장소가 할당
함수 호출과 종료에 따라 높은 주소에서 낮은 주소로 메모리가 할당되었다가 다시 제거되는 작업이 반복

13. 구조체와 공용체

[학습목표]

▶구조체와 공용체를 이해하고 설명할 수 있다.

- 구조체의 개념과 정의 방법
- 필요한 구조체 변수의 선언 방법
- 구조체 변수의 접근연산자 .의 사용 방법
- 공용체의 정의와 변수 선언 및 활용 방법

▶자료형 재정의를 위한 typedef를 사용할 수 있다.

- 키워드 typedef를 사용한 자료형 재정의 방법과 필요성
- 구조체 정의를 새로운 자료형으로 재정의

▶구조체 포인터와 배열을 활용할 수 있다.

- 구조체의 주소를 저장하는 포인터의 선언과 활용
- 구조체 포인터의 접근연산자 ->의 사용 방법
- 구조체 배열의 선언과 활용방법

13. 구조체와 공용체

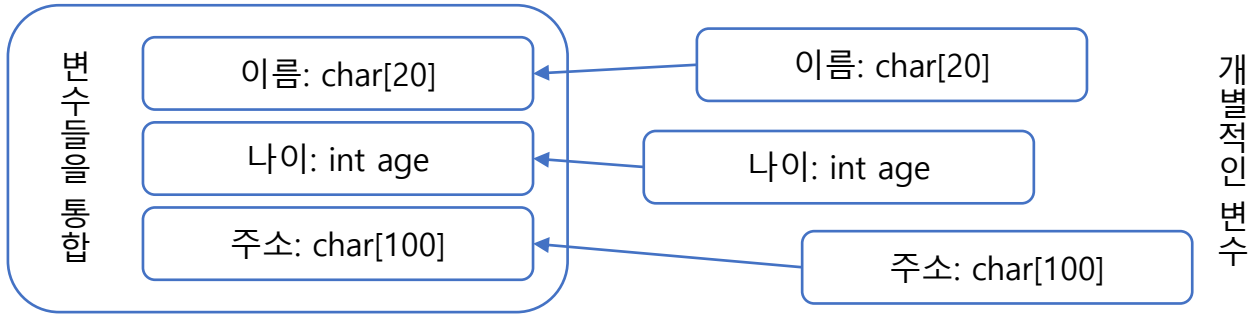
- 구조체와 공용체

구조체 개념

정수나 문자, 실수나 포인터 그리고 이들의 배열 등을 묶어 하나의 자료형으로 이용하는 것
구조체(struct)는 연관성이 있는 서로 다른 개별적인 자료형의 변수들을 하나의 단위로 묶은 새로운 자료형
구조체는 연관된 멤버로 구성되는 통합 자료형으로 대표적인 유도 자료형이다.
즉, 기존 자료형으로 새로이 만들어진 자료형을 유도 자료형이라 한다.

유도 자료형은 사용자정의 자료형으로도 불리고, 구조체, 공용체, 열거형 등이 있으며, 배열과 포인터도 유도 자료형이다.

구조체



구조체 정의

```
struct lectture
{
    char name[20];   //강좌명
    int credit;      //학점
    int hour;        //시수
};
```

struct 구조체 태그 이름
{
자료형 변수명1;
자료형 변수명2;
};

구조체 멤버 (필드)
초기값을 설정할 수 없다.
구조체 멤버로는 일반 변수, 포인터 변수, 배열,
다른 구조체 변수 및 구조체 포인터도 허용

구조체 정의는 변수의 선언과는 다른 것으로
변수선언에서 이용될 새로운 구조체 자료형을 정의하는 구문

13. 구조체와 공용체

- 구조체와 공용체

구조체 변수 선언과 초기화

```
struct account
{
    char name[20];  //계좌주이름
    int actnum;     //계좌번호
    double balance; //잔고
}myaccount;

struct account youraccount;
```

→ 구조체 정의와 변수 선언을 함께하는 문장

변수 myaccount는 struct account형 변수로 선언

struct 구조체태그이름 변수명;
struct 구조체채그이름 변수명1, 변수명2, 변수명3, … ;

```
struct
{
    char name[20];  //계좌주이름
    int actnum;     //계좌번호
    double balance; //잔고
}myaccount;
```

→ **이름 없는 구조체**

구조체 태그이름을 생략할 수 있다.
태그이름이 없는 구조체 정의에서는 바로 변수가 나오지 않으면 아무 의미 없는 문장이다.

변수 myaccount이후에 이와 동일한 구조체의 변수선언은 불가능하다.

```
struct account mine = { "홍길동", 1001, 3000000 };
```

구조체 변수의 초기화
struct 구조체태그이름 변수명 = {초기값1, 초기값2, …};
초기화 값은 중괄호 내부에서 구조체의 각 멤버 정의 순서대로 쉼표로 구분하여 기술
배열과 같이 초기값에 기술되지 않은 멤버값은 자료형에 기본값(0)이 저장

```
구조체변수이름.멤버
mine.action = 1002; mine.balance = 3000000;
```

→ **구조체 멤버 전근 연산자**

선언된 멤버 접근 연산자 .와 변수 크기

13. 구조체와 공용체

- 구조체와 공용체

구조체 활용

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

```
struct account
{
    struct date open; //계좌 개설일자
    char name[20];    //계좌주이름
    int actnum;       //계좌번호
    double balance;   //잔고
}myaccount;

struct account youraccount;
struct account mine = { {2012, 3, 9}, "홍길동", 1001, 3000000 };
```

→ **구조체 멤버로 사용되는 구조체**

자료형 accoun변수 mine

13. 구조체와 공용체

- 구조체와 공용체

공용체 개념

동일한 저장 장소에 여러 자료형을 저장하는 방법으로, 공용체를 구성하는 멤버에 한 번에 한 종류만 저장하고 참조할 수 있다.
공용체는 서로 다른 자료형의 값을 동일한 저장공간에 저장하는 자료형이다.

```c
#include <stdio.h>

//유니온 구조체를 정의하면서 변수 data1도 선언한 문장
union data
{
    char ch;     //문자형
    int cnt;     //정수형
    double real;//실수형
} data1;         //data1은 전역변수

int main(void) {
    union data data2 = { 'A' }; //첫 멤버인 char 형으로만 초기화 가능
    union data data3 = data2; //다른 변수로 초기화 기능

    printf("%d %d₩n", sizeof(union data), sizeof(data3));

    //멤버 ch에 저장
    data1.ch = 'a';
    printf("%c %d %lf₩n", data1.ch, data1.cnt, data1.real);
    //멤버 cnt에 저장
    data1.cnt = 100;
    printf("%c %d %lf₩n", data1.ch, data1.cnt, data1.real);
    //멤버 real에 저장
    data1.real = 3.156759;
    printf("%c %d %lf₩n", data1.ch, data1.cnt, data1.real);

    return 0;
}
```

공용체 구성요소인 멤버이다.

구조체선언 방법과 동일

union 공용체 태그 이름
{
자료형 변수명1;
자료형 변수명2;
};

공용체 멤버 접근을 위해 전근연산자 .를 사용

13. 구조체와 공용체

- 자료형 재정의

자료형 재정의 typedef

Typedef는 이미 사용되는 자료 유형을 다른 새로운 자료형 이름으로 재정의할 수 있도록 하는 키워드이다.

```c
#include <stdio.h>

//함수 외부에서 정의된 자료형은 이후 파일에서 사용 가능
typedef unsigned int budget;

int main(void)
{
    //새로운 자료형 budget 사용
    budget year = 24500000;

    //함수 내부에서 정의된 자료형은 이후 함수내부에서만 사용 가능
    typedef int profit;

    //새로운 자료형 profit 사용
    profit month = 4600000;

    printf("올 예산은 %d, 이달의 이익은 %d 입니다.₩n", year, month);

    return 0;
}

void test(void)
{
    //새로운 자료형 budget 사용
    budget year = 24500000;

    //profit은 이 함수에서는 사용 불가, 오류 발생
    //profit year;
}
```

[자료형 재정의 구문]
typedef 기존자료유형이름 새로운자료형1, 새로운자료형2, ... ;

unsigned year과 동일

int month와 동일

일반적으로 자료형을 재정의하는 이유는
프로그램 시스템 간 호환성과 편의성을 위해 필요

13. 구조체와 공용체

- 자료형 재정의

구조체 자료형 재정의

typedef 사용하여 구조체 struct date가 정의된 상태에서 date로 재정의할 수 있다.

```c
#include <stdio.h>

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

//struct date 유형을 간단히 date형으로 사용하기 위한 구문
typedef struct date date; //재정의

int main(void)
{
    //구조체를 정의하면서 바로 자료형 software로 정의하기 위한 구문
    typedef struct
    {
        char title[30];     //제목
        char company[30];   //제작회사
        char kinds[30];     //종류
        date release;       //출시일
    } software;

    software vs = { "비주얼 스튜디오 커뮤니티", "MS", "통합개발환경", {2020, 8, 29} };

    printf("제품명: %s₩n", vs.title);
    printf("회사: %s₩n", vs.company);
    printf("종류: %s₩n", vs.kinds);
    printf("출시일: %d. %d. %d₩n", vs.release.year, vs.release.month, vs.release.day);

    return 0;
}
```

자료유형인 date는 struct date와 함께 동일한 자료유형으로 이용이 가능

typedef로 인해 struct date가 하나의 자료형이 되었음을 보여주는 예

software는 변수가 아니라 구조체의 새로운 자료형이다.

13. 구조체와 공용체

- 자료형 재정의

구조체 포인터

포인터는 각각의 자료형 저장 공간의 주소를 저장하듯이 구조체 포인터는 구조체의 주소값을 저장할 수 있는 변수이다.
구조체 포인터 변수의 선언은 일반 포인터 변수 선언과 동일하다.

```c
#include <stdio.h>

struct lecture
{
    char name[20];  //강좌명
    int type;   //강좌구분 0: 교양, 1: 일반선택, 2: 전공필수, 3: 전공선택
    int credit; //학점
    int hours; //시수
};
typedef struct lecture lecture; //재정의

//제목을 위한 문자열
char* head[] = { "강좌명", "강좌구분", "학점","시수" };
//강좌 종류를 위한 문자열
char* lectype[] = { "교양", "일반선택", "전공필수", "전공선택" };

int main(void)
{
    lecture os = { "운영체제",2,3,3 };
    lecture c = { "C프로그래밍",3,3,4 };
    lecture* p = &os;

    printf("구조체크기 : %d, 포인터크기 : %d\n", sizeof(os), sizeof(p));
    printf("%10s %12s %6s %6s\n", head[0], head[1], head[2], head[3]);
    printf("%12s %10s %5d %5d\n", p->name, lectype[p->type], p->credit, p->hours);

    //포인터 변경
    p = &c;
    printf("%12s %10s %5d %5d\n", (*p).name, lectype[(*p).type], (*p).credit, (*p).hours);
    printf("%12c %10s %5d %5d\n", *c.name, lectype[c.type], c.credit, c.hours);

    return 0;
}
```

자료유형 lecture은 struct lecture와 동일한 자료유형으로 이용 가능

//구조체 포인터 변수 p 선언

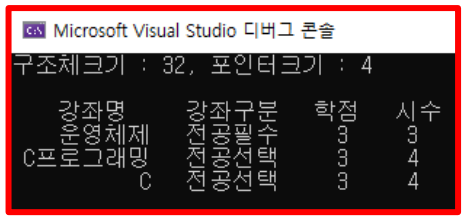
운영체제, lectype[2], 3, 3

포인터 변수의 구조체 멤버 접근 연산자 ->

p->name은 포인터 p가 가리키는 구조체 변수의 멤버 name을 접근하는 연산식이다.

- -와 >사이에 공백이 들어가서는 안된다.
- **접근연산자(.)가 간접연산자(*)보다 우선순위가 빠르다.**

접근연산식	구조체 변수 os와 구조체 포인터변수 p인 경우의 의미
p->name	포인터 p가 가리키는 구조체의 멤버 name
(*p).name	포인터 p가 가리키는 구조체의 멤버 name
*p.name	*(p.name)이고 p가 포인터이므로 p.name은 문법오류 발생
*os.name	*(os.name)를 의미, 구조체 변수os의 멤버 포인터 name이 가리키는 변수로, 이 경우는 구조체 변수 os 멤버 강좌명의 첫 문자.
*p->name	*(p->name)을 의미, 포인터 p이 가리키는 구조체의 멤버 name이 가리키는 변수로 이 경우는 구조체 포인터 p이 가리키는 구조체의 멤버 강좌명의 첫 문자.



13. 구조체와 공용체

- 자료형 재정의

구조체 배열

다른 배열과 같이 동일한 구조체 변수가 여러 개 필요하면 구조체 배열을 선언하여 이용할 수 있다.

```c
#include <stdio.h>

struct lecture
{
    char name[20];
    int type;
    int credit;
    int hours;
};
typedef struct lecture lecture;

char* lectype[] = { "교양","일반선택","전공필수","전공선택" };
char* head[] = { "강좌명","강조구문","학점","시수" };

int main(void)
{
    //구조체 lecture의 배열 선언 및 초기화
    lecture course[] = { {"인간과 사회",0,2,2},
    {"경제학개론", 1,3,3},
    {"자료구조",2,3,3},
    {"모바일프로그래밍",2,3,4},
    {"고급 C프로그래밍",3,3,4}
    };

    int arysize = sizeof(course) / sizeof(course[0]);

    printf("배열크기: %d₩n₩n", arysize);

    printf("%12s      %12s %6s %6s₩n", head[0], head[1], head[2], head[3]);
    printf("=============================================₩n");
    for (int i = 0; i < arysize; i++)
        printf("%16s %10s %5d %5d₩n", course[i].name, lectype[course[i].type], course[i].credit, course[i].hours);

    return 0;
}
```

course[0].name ="인간과 사회"	course[0].type =0	course[0].credit =2	course[0].hours =2
course[1].name ="경제학개론"	course[1].type =1	course[1].credit =3	course[1].hours =3
course[2].name ="자료구조"	course[2].type =2	course[2].credit =3	course[2].hours =3
course[3].name ="모바일프로그래밍"	course[3].type =2	course[3].credit =3	course[3].hours =4
course[4].name ="고급C프로그래밍"	course[4].type =3	course[4].credit =3	course[4].hours =4

Microsoft Visual Studio 디버그 콘솔

```
배열크기: 5

        강좌명       강조구문   학점   시수
=============================================
      인간과 사회         교양     2     2
      경제학개론       일반선택     3     3
        자료구조       전공필수     3     3
  모바일프로그래밍       전공필수     3     4
  고급 C프로그래밍       전공선택     3     4
```

마무리

- 이번 단원은 1학년 2학기 때 배웠던 내용이어서 친근하게 다가왔습니다. 당시 비슷하게 생긴 함수와 구조체, 공용체의 차이 등에 관련해 많이 헷갈려 했는데 그 부족함을 채워주는 계기가 되었습니다.

- 11장. 문자와 문자열에서 getche, getchar, getch, get, put, 등 함수명과 형태, 사용법도 비슷해 '어떠한 상황에 무엇을 사용하는 것이 편한가?'라는 생각이 먼저 들었던 단원이었습니다. 이를 알아가기 위해서 예제 코드를 분석해 쉽고 빠르게 이해할 수 있었습니다. 결과창으로 자신이 사용한 각 함수의 특징을 한 눈에 알 수 있고, 빌드 중에 뜨는 오류로 문제점, 주의할 점을 알 수 있었습니다. 가장 재밌게 공부한 단원이었습니다.

- 12장에서는 변수들의 유효 범위가 있고, 외부와 내부가 또 정해져 있고, 또 사용되는 키워드 등으로 제한적인 것들이 많아, 코드작성 하면서 가장 힘들었던 부분이었습니다. 특히 전역변수와 정적 변수를 이용해 코드 빌드하는 과정에서 뜨는 오류를 해결하는 것이 가장 큰 숙제였던 단원이었습니다. 개념적으로는 이해가 빨리 갔으나, 막상 코딩을 해보니까 쉽지 않다는 생각이 들었습니다.

- 13장을 복습하면서 구조체와 공용체의 개념은 정리가 되었으나, 구조체 포인터나 멤버 접근 연산자를 이해하는 데 오랜 시간이 걸린 단원이었습니다. 그 전에 포인터를 배웠을 때도 정말 힘들게 배웠는데 그 때가 떠올랐습니다. 최대한 예제 코드의 결과를 보고 이 포인터가 무엇을 가리키고, 왜 이런 결과가 나왔는 지를 생각하면서 복습을 했습니다. 그 전에 포인터 개념을 배워서 그나마 쉽게 배웠던 것 같습니다. 나중에 C언어로 프로젝트를 만들게 된다면 구조체나 공용체를 필드 선언하는 데 많이 사용될 거 같으면서도 유용할 거 같다고 생각이 든 단원이었습니다.