
C 애플리케이션구현

학과 : 컴퓨터정보공학과
학번 : 20164107
이름 : 박민성

2020 학년도 1학기	전공	컴퓨터정보공학과(사회맞춤형 지능형 컴퓨팅과정)	학부	컴퓨터공학부
과 목 명	C에플리케이션구현(2016003-PE)			
강의실 과 강의시간	월:5(3-217),6(3-217),7(3-217),8(3-217)		학점	3
교과분류	이론/실습		시수	4
담당 교수	강환수 + 연구실 : 2호관-706 + 전 화 : 02-2610-1941 + E-MAIL : hskang@dongyang.ac.kr + 면담가능기간 : 월 11시~12시 화 14시~17시			
학과 교육목표				
과목 개요	본 과목은 프로그래밍 언어 중 가장 널리 사용되고 있는 C언어를 학습하는 과목으로 C++, JAVA 등과 같은 언어의 기반이 된다. 본 과목에서는 지난 학기에서 배운 시스템프로그래밍1에 이어 C언어의 기본 구조 및 문법 체계 그리고 응용 프로그래밍 기법 등을 다룬다. C언어에 대한 학습은 Windows상에서 이루어지며, 기본적인 이론 설명 후 실습문제를 프로그래밍하며 숙지하는 형태로 수업이 진행된다.			
학습목표 및 성취수준	대학 교육목표와 학과 교육목표를 달성하기 위하여 이 과목을 수강함으로써 학습자는 C언어의 문법 전반과 응용 프로그램 기법을 알 수 있다. 직전 학기의 수강으로 인한 C언어의 기초부터 함수, 포인터 등의 내용 이해를 바탕으로하여 이번 학기에는 지난 학기 내용의 전체적인 복습과 함께 C언어 전체를 학습하고, 특히 응용 능력을 배양하여 프로그래밍으로 문제를 해결하는 능력을 익히게 된다.			
	도서명	저자	출판사	비고
주교재	Perfect C	강환수, 강환일, 이동규	인피니티북스	
수업시 사용도구	Visual C++			
평가방법	중간고사 30%, 기말고사 30%, 과제물 및 퀴즈 20%, 출석 20%			
수강안내	C 언어를 활용하여 응용프로그램을 구현할 수 있다.			
1 주차	[개강일(3/16)]			
학습주제	강의 소개 및 전 학기 강의 내용 복습; C언어 기초 및 조건문과 반복문 복습			
목표및 내용	C언어 기초 통합개발환경 테스트 기초적인 코드 실습			

미리읽어오기	교재 1~5장
과제,시험,기타	수업 중에 제시함
2 주차	[2주]
학습주제	C언어 기초 문법
목표및 내용	변수와 상수 연산자 l-value와 r-value
미리읽어오기	교재 1~5장
과제,시험,기타	수업 중에 제시함
3 주차	[3주]
학습주제	조건문
목표및 내용	6장 조건문 학습
미리읽어오기	교재 6장
과제,시험,기타	수업 중에 제시함
4 주차	[4주]
학습주제	반복문
목표및 내용	7장 반복문 학습
미리읽어오기	교재 7장
과제,시험,기타	수업 중에 제시함
5 주차	[5주]
학습주제	포인터
목표및 내용	8장 포인터 학습 단일포인터 다중포인터 여러가지 포인터
미리읽어오기	교재 8장
과제,시험,기타	수업 중에 제시함
6 주차	[6주]
학습주제	배열
목표및 내용	9장 배열
미리읽어오기	교재 9장
과제,시험,기타	수업 중에 제시함

7 주차	[7주]
학습주제	함수
목표및 내용	10장 함수
미리읽어오기	교재 10장
과제,시험,기타	수업 중에 제시함
8 주차	[중간고사]
학습주제	중간고사
목표및 내용	중간고사
미리읽어오기	
과제,시험,기타	
9 주차	[9주]
학습주제	문자열
목표및 내용	11장 문자열
미리읽어오기	교재 11장
과제,시험,기타	수업 중에 제시함
10 주차	[10주]
학습주제	변수 유효범위
목표및 내용	12장 변수 유효범위
미리읽어오기	교재 12장
과제,시험,기타	수업 중에 제시함
11 주차	[11주]
학습주제	구조체
목표및 내용	13장 구조체
미리읽어오기	교재 13장
과제,시험,기타	수업 중에 제시함
12 주차	[12주]
학습주제	함수와 포인터 활용
목표및 내용	14장 함수와 포인터 활용
미리읽어오기	교재 14장
과제,시험,기타	수업 중에 제시함

13 주차	[13주]
학습주제	파일처리
목표및 내용	15장 파일처리
미리읽어오기	교재 15장
과제,시험,기타	수업 중에 제시함
14 주차	[14주]
학습주제	향상심화강좌(동적할당)
목표및 내용	16장 동적할당
미리읽어오기	교재 16장
과제,시험,기타	수업 중에 제시함
15 주차	[기말고사]
학습주제	기말고사
목표및 내용	기말고사
미리읽어오기	
과제,시험,기타	..
수업지원 안내	<p>장애학생을 위한 별도의 수강 지원을 받을 수 있습니다.</p> <p>언어가 문제가 되는 학생은 글로 된 과제 안내, 확대문자 시험지 제공 등의 지원을 드립니다.</p>

● 목차

1. C언어

- 1.1. C언어의 특징
- 1.2. C 언어를 배워야 하는 이유
- 1.3. 통합개발환경, 개발도구

2. 11장 문자와 문자열

- 2.1. 문자와 문자열
- 2.2. 문자열 관련함수
- 2.3. 여러 문자열 처리

3. 12장 변수 유효범위

- 3.1. 지역변수와 전역변수
- 3.2. 정적 변수와 레지스터 변수
- 3.3. 메모리 영역과 변수 이용

4. 13장 구조체와 공용체

- 4.1. 구조체와 공용체
- 4.2. 자료형 재정의
- 4.3. 구조체와 공용체의 포인터와 배열

5. 14장 함수와 포인터 활용

- 5.1. 함수의 인자전달 방식
- 5.2. 포인터 전달과 반환
- 5.3. 함수 포인터 void 포인터

6. 15장 파일처리

- 6.1. 파일 기초
- 6.2. 텍스트 파일 입출력
- 6.3. 이진 파일 입출력, 파일 접근 처리

7. 맺음말

1. C 언어

1.1 C 언어의 특징

- C 언어는 함수 중심으로 구현되는 **절차지향 언어**이다.
절차지향 언어란 시간의 흐름에 따라 정해진 절차를 실행한다는 의미로 C 언어는 문제의 해결 순서와 절차의 표현과 해결이 쉽도록 설계된 **프로그램 언어**이다.
- C 언어는 **간결하고 효율적인 언어**이다.
C 언어는 비트연산, 증감연산, 축약대입연산과 같은 다양한 연산과 이미 개발된 다양한 시스템 라이브러리를 제공하며, 함수의 재귀호출이 가능하므로 간결한 소스로 프로그램을 작성할 수 있다. 또한 운영체제 유닉스 사스 시스템을 개발하기 위한 목적으로 고안된 언어로 시스템의 세세한 부분까지 제어할 수 있도록, 포인터와 메모리 관리 기능을 갖고있으며, 여러 뛰어난 기능으로 다양한 분야에서 널리 활용되고 있다.
- C 언어로 작성된 **프로그램은 크기도 작으며**, 메모리도 적게 효율적으로 사용하여 **실행 속도가 빠르다**.
- C 언어는 어셈블리 언어로 만든 장점뿐만 아니라 다시 개발하는 일이 없이, 간단히 컴파일만 다시 하여 원하는 결과를 얻을수 있어 다양한 CPU와 플랫폼의 컴파일러를 지원하기 때문에 **이식성이 좋다**.

1.2 C 언어를 배워야 하는 이유

- 현장에서 많이 쓰이는 자바나 C#, Objective-C 등의 뿌리가 C이므로, C언어를 알면 JAVA나 C#, Objective-C 뿐만 아니라 그 이후의 프로그래밍 언어들은 가지를 뺀어 나가듯 습득이 쉬워진다.
- C 언어는 범용적인 프로그래밍 언어로 임베디드 시스템에서부터 응용 프로그램운영체제와 같은 시스템 소프트웨어 개발 등 여러 분야에 널리 사용된다.

1.3 통합개발환경, 개발도구

- **마이크로소프트 비주얼 스튜디오**
: 비주얼 스튜디오는 프로그램 언어 C/C++ 뿐만 아니라 C#, JavaScript, Python, Visuam Basic 등의 여러 프로그램 언어를 이용하여 응용 프로그램 및 앱을 개발할 수 있는 다중 플랫폼 개발 도구이며, 윈도우 운영체제에서 이용되고 있다.
- **이클립스 C/C++ IDE**
: 이클립스는 이클립스 컨소시엄이 개발한 유니버설 도구 플랫폼으로 모든 부분에 대해 개방형이며 PDF환경을 지원하여 확장 가능한 통합개발환경이다. C/C++ 개발자용 IDE가 C/C++를 개발하기 위한 개발도구로 컴파일러는 따로 설치해야 한다.

2. 11장 문자와 문자열

2.1 문자와 문자열

- 문자와 문자열의 개념

: 문자는 영어의 알파벳이나 한글의 한 글자를 **작은 따옴표**로 둘러싸서 'A'와 같이 표기하며, C 언어에서 저장공간 크기 1바이트인 자료형 char로 지원한다. 작은 따옴표에 의해 표기된 문자를 문자 상수라 한다.

ex) char ch = 'A';

문자의 모임인 일련의 문자를 문자열이라 한다. 이러한 문자열은 일련의 문자 앞뒤로 **큰 따옴표**로 둘러싸서 "java"로 표기한다.

문자의 나열인 문자열은 'ABC'처럼 작은 따옴표로 둘러싸도 문자가 될 수 없으며 오류가 발생한다.

ex) char c[] = "C language";

- 문자와 문자열의 선언

: C 언어에서 char형 변수에 문자를 저장한다. 그러나 C언어는 문자열을 저장하기 위한 자료형을 따로 제공하지 않는다. 문자열을 저장하려면 문자의 모임인 '문자 배열'을 사용한다.

문자열의 마지막을 의미하는 NULL 문자 '\0'가 마지막에 저장 되어야한다.

배열 선언 시 초기화 방법을 이용하면 문자열을 쉽게 저장할 수 있다. 주의할 점은 중괄호를 사용하게 되며, 문자 하나 하나를 쉼표로 구분하여 입력하고 마지막 문자로 NULL인 '\0'을 삽입해야 한다.

ex) char java[] = {'J','A','V','A','\0'};

문자열을 선언하는 편리한 다른 방법으로 배열 선언 시 저장할 큰 따옴표를 사용해 문자열 상수를 바로 대입하는 방법이다.

■ 배열 초기화 시 배열크기는 지정하지 않는 것이 더 편리

■ 지정한다면 실제 문자 수보다 1이 더크게 배열크기를 지정해야 한다

■ 만일 지정한 배열크기가 (문자수+1)보다 크면 나머지 부분은 모두 '\0'문자로 채워진다.

ex) char c[] = "C language";

ex) char c[11] = "C language";

- printf()를 이용한 문자와 문자열 출력

: 함수 printf()에서 형식제어문자 %c로 문자를 출력한다. 함수 printf()에서 배열이름 또는 문자포인터를 사용하여 형식제어문자 %s로 문자열을 출력한다. 함수 puts(csharp)와 같이 사용하면 한중에 문자열을 출력한 후 다음 중에서 출력을 준비한다. 함수 printf(c)와 같이 바로 배열이름을 인자로 사용해도 문자열 출력이 가능하다.

*** 예제코드 chararray.c**

문자열 저장을 위한 문자열 배열 처리와 문자열 출력

```
#include <stdio.h>

int main(void)
{
    char ch = 'a';
    printf("%c %d\n", ch, ch);

    char java[] = { 'j', 'a', 'v', 'a', '\0' };
    printf("%s\n", java);

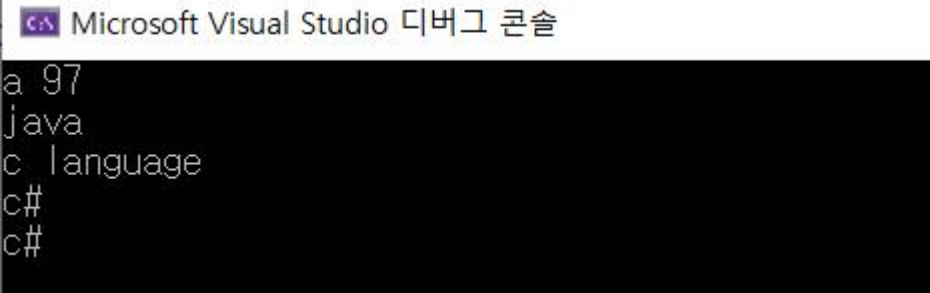
    char c[] = "c language";
    printf("%s\n", c);

    char csharp[5] = "c#";
    printf("%s\n", csharp);

    printf("%c%c\n", csharp[0], csharp[1]);

    return 0;
}
```

*** chararray.c 실행결과**



```
Microsoft Visual Studio 디버그 콘솔
a 97
java
c language
c#
c#
```

- 문자열 구성하는 문자 참조

: 문자열을 처리하는 다른 방법은 문자열 상수를 문자 포인터에 저장하는 방식이다.
문자포인터 변수에 문자열 상수를 지정할 수 있다.

ex) `int i = 0;`
`char *java = "java";`

문자열 출력도 함수 printf()에서 포인터 변수와 형식제어문자 %s로 간단히 처리할 수 있다. ex) printf("%s", java);

문자열을 구성하는 하나 하나의 문자를 배열형식으로 직접 참조하여 출력하는 방식도 있다. 그러나 변수 java를 사용하여 문자를 수정하거나 수정될 수 있는 함수의 인자로 사용하면 오류가 발생한다.

- 출력할 문자열의 끝을 '\0'문자로 검사하면 편리
- 문자열 상수를 저장하는 문자 포인터는 사용상 주의가 필요

*** 예제코드 charpointer.c**

문자 포인터로 문자열 처리

```
#include <stdio.h>

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

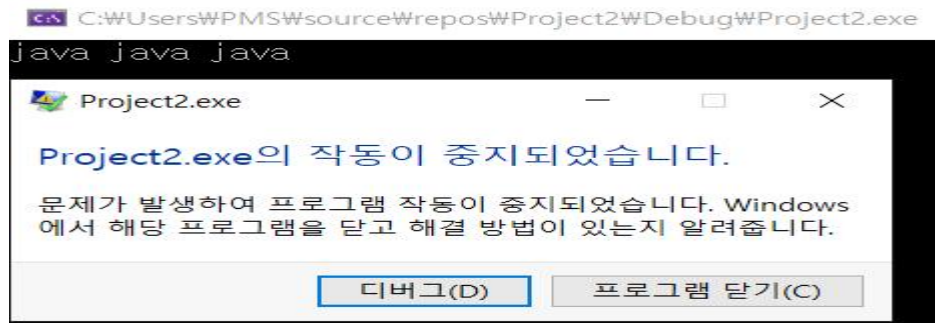
    int i = 0;
    while (java[i])
        printf("%c", java[i++]);
    printf(" ");

    i = 0;
    while (*(java + i) != '\0')
        printf("%c", *(java + i++));
    printf("\n");

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

    return 0;
}
```

* charpointer.c 실행결과



java[0] = 'J'; 코드로 인해 오류 발생

- 'w0'문자에 의한 문자열 분리

: 함수 printf()에서 %s는 문자 포인터가 가리키는 위치에서 NULL 문자까지를 하나의 문자열로 인식한다.

```
char c[] = "C C++ Java";
c[5] = '\0';
printf("%s\n%s\n", c, (c+6));
```

위 소스에서 배열c[]는 처음에 문자열 "C C++ Java"가 저장되고 마지막에 NULL 문자가 저장된다.

-소스 실행결과-

```
C C++
Java
```

* 예제코드string.c

문자포인터로 문자열처리

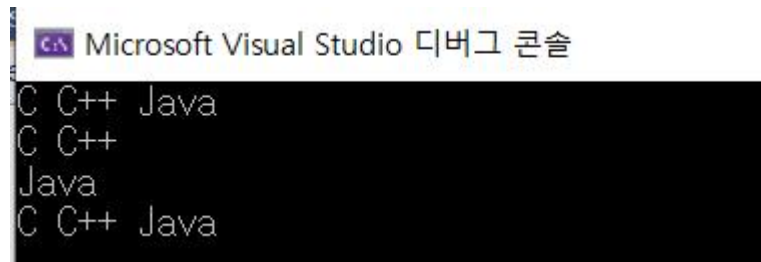
```
#include <stdio.h>

int main(void) {
    char c[] = "C C++ Java";
    printf("%s\n", c);
    c[5] = '\0';
    printf("%s\n%s\n", c, (c + 6));

    c[5] = ' ';
    char* p = c;
    while (*p)
        printf("%c", *p++);
    printf("\n");

    return 0;
}
```

* **string.c 실행결과**



```

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

```

- **다양한 문자 입출력**

- 버퍼처리 함수 `getchar()`
 : 문자의 입력에 사용되고 `putchar()`는 문자의 출력에 사용된다.
 문자 입력을 위한 함수 `getchar()`는 라인 버퍼링 방식을 사용
 입력한 문자는 임시 저장소인 버퍼에 저장되었다가 [Enter] 키를 만나면
 버퍼에서 문자를 읽기 시작한다.
- 함수 `getche()`
 : 입력된 문자는 바로 모니터에 나타난다.
 버퍼를 사용하지 않고 문자 하나를 바로 바로 입력할 수 있는 함수
 헤더파일 `conio.h` 삽입 필요
- 함수 `getch()`
 : 입력한 문자가 화면에 보이지 않는 특성이있다.
 버퍼를 사용하지 않는 문자 입력 함수
 헤더파일 `conio.h` 삽입 필요
- 문자입력 함수 비교

함수	<code>scanf("%c", &ch)</code>	<code>getchar()</code>	<code>getche()</code> <code>_getche()</code>	<code>getch()</code> <code>_getch()</code>
헤더파일	stdio.h		conio.h	
버퍼 이용	버퍼 이용함		버퍼 이용 안함	
반응	[Enter] 키를 눌러야 반응		문자 입력마다 반응	
입력 문자의 표시 (echo)	누르면 바로 표시		누르면 바로 표시	표시 안됨
입력문자 수정	가능		불가능	

* 예제코드 getche.c

함수 getchar(), _getche(), _getch()의 차이를 알아보는 코드

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

int main(void) {
    char ch;

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

    printf("\n문자를 누를 때마다 두 번 출력 >>\n");
    while ((ch = _getche()) != 'q')
        putchar(ch);

    printf("\n문자를 누르면 한 번 출력 >>\n");
    while ((ch = _getch()) != 'q')
        _putch(ch);
    printf("\n");

    return 0;
}
```

* getche.c 실행결과

C:\Users\WPMSW\source\repos\Project2\Debug\Project2.exe

문자를 계속 입력하고 Enter를 누르면 >>

java
java
park
park
min
min
q

문자를 누를 때마다 두 번 출력 >>

jjaavvaag

문자를 누르면 한 번 출력 >>

javaaparkminsung

- 문자열 입력

- scanf()

: 공백으로 구분되는 하나의 문자열을 입력받을 수 있다.

함수 scanf(:"%s", str)에서 형식제어문자 %s를 사용하여 문자열을 입력받음

* 예제코드 **stringput.c**

함수 scanf()와 printf()에서의 문자열 입출력

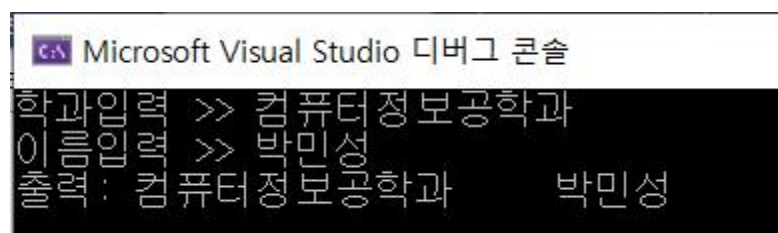
```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void) {
    char name[20], dept[30];

    printf("%s", "학과입력 >> ");
    scanf("%s", dept);
    printf("%s", "이름입력 >> ");
    scanf("%s", name);
    printf("출력: %10s %10s\n", dept, name);

    return 0;
}
```

* **stringput.c** 실행결과



```
C:\N Microsoft Visual Studio 디버그 콘솔
학과입력 >> 컴퓨터정보공학과
이름입력 >> 박민성
출력: 컴퓨터정보공학과      박민성
```

- gets()와 puts()

: 함수 gets()는 한 행의 문자열 입력에 유용한 함수

함수 puts()는 한 행에 문자열을 출력하는 함수

함수 gets(), puts(), gets_s()를 사용하려면 헤더파일 stdio.h 삽입 필요

```
gets()   : char * gets(char * buffer);
gets_s() : char * gets_s(char * buffer, size_t sizebuffer);
puts()   : int puts(const char * str);
```

* 예제코드 gets.c

gets()와 gets_s()를 사용하여 여러 줄을 입력 받아 출력하는 프로그램
while문을 사용하면 연속된 여러 행을 입력 받아 바로 행 별로 출력할 수
있다. 다음 반복을 종료하려면 새로운 행 처음에 (ctrl + Z)를 입력한다.
함수 printf()와 scanf()는 다양한 입출력에 적합하며, 문자열 입출력 함수
put()와 gets()ss 처리 속도가 빠르다는 장점이 있다.

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

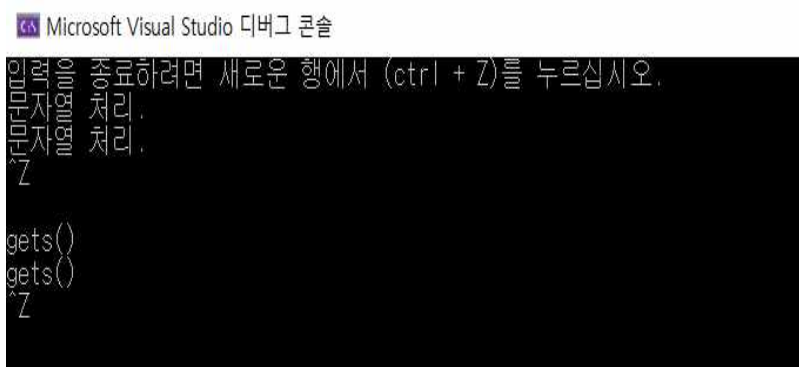
int main(void) {
    char line[101];

    printf("입력을 종료하려면 새로운 행에서 (ctrl + Z)를 누르십시오.\n");
    while (gets(line))
        puts(line);
    printf("\n");

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

    return 0;
}
```

* gets.c 실행결과



```
Microsoft Visual Studio 디버그 콘솔
입력을 종료하려면 새로운 행에서 (ctrl + Z)를 누르십시오.
Z
gets()
gets()
Z
```


2.2 문자열 관련 함수

- 문자열 라이브러리 함수

: 문자의 배열 관련 함수는 헤더파일 **string.h**에 함수원형이 정의되어 있다.

함수원형	설명
void *memchr(const void *str, int c, size_t n)	메모리 str에서 n 바이트까지 문자 c를 찾아 그 위치를 반환
int memcmp(const void *str1, const void *str2, size_t n)	메모리 str1과 str2를 첫 n 바이트를 비교 검색하여 같으면 0, 다르면 음수 또는 양수 반환
void *memcpy(void *dest, const void *src, size_t n)	포인터 src 위치에서 dest에 n 바이트를 복사한 후 dest위치 반환
void *memmove(void *dest, const void *src, size_t n)	포인터 src 위치에서 dest에 n 바이트를 복사한 후 dest위치 반환
void *memset(void *str, int c, size_t n)	포인터 str 위치에서부터 n 바이트까지 문자 c를 지정 한 후 str 위치반환
size_t strlen(const char *str)	포인터 str 위치에서부터 널 문자를 제외한 길이 반환

* 예제코드 memfun.c

문자열의 길이를 반환하는 함수 strlen()과 문자배열의 복사를 위한 함수 memcpy(), 그리고 문자배열에서 문자 이후의 문자열을 찾는 함수 memchr()을 알아보는 예제

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

int main(void) {
    char src[50] = "https://www.visualstudio.com";
    char dst[50];

    printf("문자배열 src = %s\n", src);
    printf("문자열크기 strlen(src) = %d\n", strlen(src));
    memcpy(dst, src, strlen(src) + 1);
    printf("문자배열 dst = %s\n", dst);
    memcpy(src, "안녕하세요!", strlen("안녕하세요!") + 1);
    printf("문자배열 src = %s\n", src);

    char ch = ':';
    char* ret;
    ret = memchr(dst, ch, strlen(dst));
    printf("문자 %c 뒤에는 문자열 %s 이 있다.\n", ch, ret);

    return 0;
}
```


* memfun.c 실행결과

Microsoft Visual Studio 디버그 콘솔

```
문자배열 src = https://www.visualstudio.com
문자열 크기 strlen(src) = 28
문자배열 dst = https://www.visualstudio.com
문자배열 src = 안녕하세요!
문자 : 뒤에는 문자열 ://www.visualstudio.com 이 있다.
```

- strcmp(), strncmp()

: strcmp()는 인자인 두 문자열을 사전상의 순서로 비교하는 함수
= `int strcmp(const char * s1, const char * s2);`

strncmp()는 두 문자를 비교할 문자의 최대 수를 지정하는 함수
= `int strncmp(const char * s1, const char *s2, rsize_t maxn);`

* 예제코드 strcmp.c

문자열 비교함수 strcmp()와 strncmp()사용예제

```
#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;
}
```

* strcmp.c 실행결과

Microsoft Visual Studio 디버그 콘솔

```
strcmp(java, java) = 0
strcmp(java, jav) = 1
strcmp(jav, java) = -1
strncmp(jav, java, 3) = 0
```

- **strcpy(), strncpy()**

: 문자열을 복사하는 함수

strcpy()는 앞 인자 문자열 dest에 뒤 인자 문자열 source를 복사한다.

* **예제코드 strcpy.c**

문자열 복사 함수 strcpy()와 strncpy()의 사용 예제코드

```
#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.";

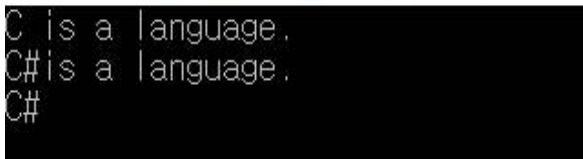
    printf("%s\n", strcpy(dest, source));
    printf("%s\n", strncpy(dest, "C#", 2));

    printf("%s\n", strncpy(dest, "C#", 3));

    return 0;
}
```

* **strcpy.c 실행결과**

 Microsoft Visual Studio 디버그 콘솔



-> "C#w0"까지 3개의 문자가 복사
되므로 문자열 "C#"이 출력된다.

- **strcat(), strncat()**

: strcat()는 앞 문자열에 뒤 문자열의 null 문자까지 연결하여, 앞의 문자열 주소를 반환하는 함수

strncat()는 전달인자의 마지막에 연결되는 문자의 수를 지정하며 그 이상은 연결되지 않도록 한다.

* **예제코드 strcat.c**

문자열 연결함수 사용 예제

```

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

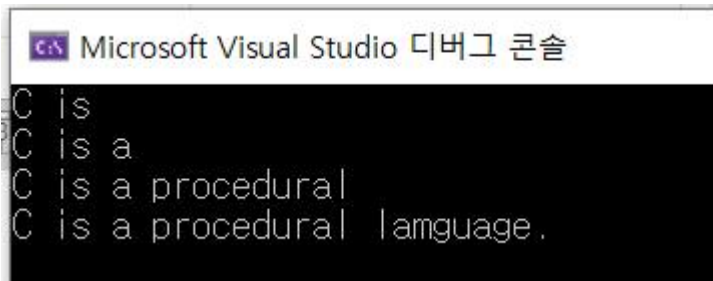
int main(void) {
    char dest[80] = "C";

    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;
}

```

* strcat.c 실행결과



```

Microsoft Visual Studio 디버그 콘솔
C is
C is a
C is a procedural
C is a procedural language.

```

- strtok(), strtok_s()

: strtok()은 문자열에서 구분자인 문자를 여러 개 지정하여 토큰을 추출하는 함수
 첫 번째 인자인 str은 토큰을 추출할 대상인 문자열이며 두 번째 인자인 delim은
 구분자로 문자의 모임인 문자열이다.

<strtok()의 사용방법>

- ▣ 문장 ptoken = strtok(str, delimiter);으로 첫 토큰을 추출한다.
- ▣ 결과를 저장한 ptoken이 NULL이면 더 이상 분리할 토큰이 없는 경우이다.
- ▣ 계속 토큰을 추출하려면 while 반복으로 추출된 토큰이 있는지를 (ptoken != NULL)로 검사하고 NULL을 첫 번째 인자로 다시 strtok(NULL, delimiter)를 호출하면 그 다음 토큰을 반환받을 수 있다.

* 예제코드 strtok.c

문자열에서 지정한 분리자를 사용하여 토큰을 사용하는 예제코드

```

#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* proken = strtok(str1, delimiter);

    while (proken != NULL) {
        printf("%s\n", proken);
        proken = strtok(NULL, delimiter);
    }
    return 0;
}

```

* strtok.c 실행결과

Microsoft Visual Studio 디버그 콘솔

```

문자열 "C and C++\t language are best!"을 >>
구분자[ ,\t!]를 이용하여 토큰을 추출 >>
C
and
C++
language
are
best

```

- 문자열의 길이와 위치 검색

: 함수 `strlen()`은 NULL 문자를 제외한 문자열 길이를 반환하는 함수
`strlwr()`는 인자를 모두 소문자로 변환하여 반환
`strupr()`는 인자를 모두 대소문자로 변환하여 반환

<문자열 관련 함수>

```

char * strlwr(char * str);
errno_t _strlwr_s(char *str, rsize_t strsize);

```

문자열 `str`을 모두 소문자로 변환하고 변환한 문자열을 반환하므로 `str`은 상수이면 오류가 발생하며, `errno_t`는 정수형의 오류번호이며, `size_t`도 정수형으로 `strsize`는 `str`의 길이

```
char * strrpr(char * str);  
errno_t strupr_s(char * str, size_t strsize);
```

문자열 str을 모두 대문자로 변환하고 변환한 문자열을 반환하므로 str은 상수이면 오류가 발생하며, errno_t는 정수형의 오류번호이며, size_t도 정수형으로 strsize는 str의 길이

* 예제코드 strfun.c

```
#define _CRT_SECURE_NO_WARNINGS  
#include <stdio.h>  
#include <string.h>  
  
int main(void) {  
    char str[] = "JAVA 2017 go c#";  
    printf("%d\n", strlen("java"));  
    printf("%s, ", _strlwr(str));  
    printf("%s\n", _strupr(str));  
  
    printf("%s, ", strstr(str, "VA"));  
    printf("%s\n", strchr(str, 'A'));  
  
    return 0;  
}
```

* strfun.c 실행결과

Microsoft Visual Studio 디버그 콘솔

```
4  
java 2017 go c#, JAVA 2017 GO C#  
VA 2017 GO C#, AVA 2017 GO C#
```

2.3 여러 문자열 처리

- 문자 포인터 배열

```
= char *pa[] = {"JAVA", "C#", "C++"};
```

- 이차원 문자 배열

= `char ca[][5] = {"JAVA", "C#", "C++"};`

* 예제코드 strarray.c

여러 개의 문자열을 선언과 동시에 저장하고 처리하는 방법 예제

```
#include <stdio.h>

int main(void) {
    char* pa[] = { "JAVA", "C#", "C++" };
    char ca[ ][5] = { "JAVA", "C#", "C++" };

    printf("%s ", pa[0]); printf("%s ", pa[1]); printf("%s\n", pa[2]);
    printf("%s ", ca[0]); printf("%s ", ca[1]); printf("%s\n", ca[2]);

    printf("%c %c %c\n", pa[0][1], pa[1][1], pa[2][1]);
    printf("%c %c %c\n", ca[0][1], ca[1][1], ca[2][1]);

    return 0;
}
```

* strarray.c 실행결과

Microsoft Visual Studio 디버그 콘솔

```
JAVA C# C++
JAVA C# C++
A # +
A # +
```

- 명령행 인자

- `main(int argc, char *argv[])`

: 프로그램에서 명령행 인자를 받으려면 `main()` 함수에서 두 개의 인자 `argc`와 `argv`를 (`int argc, char *argv[]`)로 기술해야 한다.

매개변수 `argc`는 명령행에서 입력한 문자열의 수이며 `argv[]`는 명령행에서 입력한 문자열을 전달받는 문자 포인터 배열이다.

* 예제코드 commandarg.c

명령행 인자 출력 예제

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    int i = 0;
```



```

printf("실행 명령행 인자(command line arguments) >>\n");
printf("argc = %d\n", argc);
for (i = 0; i < argc; i++)
    printf("argv[%d] = %s\n", i, argv[i]);

return 0;
}

```

* commandarg.c 실행결과

Microsoft Visual Studio 디버그 콘솔

```

실행 명령행 인자(command line arguments) >>
argc = 1
argv[0] = C:\Users\PMS\source\repos\Project2\Debug\Project2.exe

```

2.4 프로그래밍 연습

- 한 행을 표준입력으로 입력 받은 문장의 길이를 구하는 함수 `mystrlen()`을 구현하여 라이브러리 `strlen()`과 결과를 비교하는 프로그램

```

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

int main() {
    char get[100];

    gets(get);

    printf("strlen: %d\n", strlen(get));
    printf("mystrlen: %d", mystrlen(get));
}

int mystrlen(const char *p) {
    int count = 0;
    while (p[count] != NULL)
        count++;
    return count;
}

```

Microsoft Visual Studio 디버그 콘솔

```

parkminsung
strlen: 11
mystrlen: 11

```

- 라이브러리 함수 gets()로 표준입력 받은 두 문자열을 연결하여 출력하는 프로그램

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

int main() {
    void mystcat(char s1[], const char s2[]);

    printf("문자열입력: ");
    char s1[100];
    gets(s1);
    printf("연결할 문자열입력: ");
    char s2[100];
    gets(s2);

    mystcat(s1, s2);
}

void mystcat(char s1[], const char s2[]) {
    int s1count = strlen(s1);

    for (int i = 0; s2[i] != NULL; i++) {
        s1[s1count] = s2[i];
        s1count++;
    }

    for (int i = 0; i < s1count; i++) {
        printf("%c", s1[i]);
    }
}
```

 Microsoft Visual Studio 디버그 콘솔

```
문자열입력: park
연결할 문자열입력: minsung
parkminsung
```


- 라이브러리 gets()와 scanf()로 표준입력을 받은 하나의 문자열과 문자를 사용하여 문자를 삭제하여 출력하는 프로그램

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

int main() {
    void delchar(char str[], const char ch);

    char a[100];
    char b;

    printf("문자열입력:");
    gets(a);

    printf("삭제할 문자입력:");
    scanf("%c", &b);

    delchar(a,b);
    printf("결과:%s", a);
}

void delchar(char str[], const char ch) {
    for (int i = 0; str[i] != NULL; i++) {
        if (str[i] == ch) {
            do {
                for (int z = i; str[z] != NULL; z++) {
                    str[z] = str[z + 1];
                }
            } while (str[i] == ch);
        }
    }
}
```

 Microsoft Visual Studio 디버그 콘솔

```
문자열입력:parkminsung
삭제할 문자입력:n
결과:parkmisug
```

- 문자를 입력받아 아스키 코드값을 출력하는 프로그램

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

int main() {
    char ch;
    printf("문자입력: ");
    scanf("%c", &ch);
    printf("%c의 아스키코드값은 %d입니다.\n", ch, ch);
}
```

Microsoft Visual Studio 디버그 콘솔

문자입력: T
T의 아스키코드값은 84입니다.

- 여러 줄의 문자열을 표준입력으로 입력받아 구두점의 수를 구하여 출력하는 프로그램

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

int main(void) {
    char line[100][100];

    int i = 0;
    int count = 0;

    printf("여러줄의 문자열을 입력받아 구두점의 수를 출력하는 프로그램입니다.\n");
    printf("입력을 종료하시려면 새로운 행에서 ctrl+Z 를 눌러주세요.\n");
    while (gets(line[i]))
    {
        i++;
    }

    for (int z=0; z<3; z++){
        for (int x = 0; line[z][x] != NULL; x++) {
            if (ispunct(line[z][x]))
                count++;
        }
    }

    printf("구두점의 수: %d", count);
}
```

Microsoft Visual Studio 디버그 콘솔

여러줄의 문자열을 입력받아 구두점의 수를 출력하는 프로그램입니다.
입력을 종료하시려면 새로운 행에서 ctrl+Z 를 눌러주세요.
park...
m/i/n
s.u.n.g
^Z
구두점의 수: 8

3. 12장 변수 유효범위

3.1 지역변수와 전역변수

- 변수 scope

: 변수의 참조가 유효한 범위를 변수의 유효범위(scope)라 한다.

변수의 유효범위는 크게 지역 유효범위와 전역 유효범위로 나눌 수 있다.

지역 유효범위는 함수 또는 블록 내부에서 선언되어 그 지역에서 변수의 참조가 가능한 범위이다.

- 지역변수

: 함수 또는 블록에서 선언된 변수

내부변수 또는 자동변수라고도 부른다.

선언 문장 이후에 함수나 블록의 내부에서만 사용이 가능

선언 후 초기화하지 않으면 쓰레기값이 저장된다.

변수가 선언된 함수 또는 블록에서 선언 문장이 실행되는 시점에서 메모리에 할당

지역변수는 선언된 부분에서 자동으로 생성되고 함수나 블록이 종료되는 순간

메모리에서 자동으로 제거된다.

지역변수가 할당되는 메모리 영역을 스택이라 한다.

* 예제코드 localvar.c

```
#include <stdio.h>

void sub(int param);

int main(void) {
    auto int n = 10;
    printf("%d\n", n);

    for (int m = 0, sum = 0; m < 3; m++) {
        sum += m;
        printf("%t%d %d\n", m, sum);
    }

    printf("%d\n", n);
    sub(20);

    return 0;
}

void sub(int param) {
    auto int local = 100;
    printf("%t%d %d\n", param, local);
}
```

* localvar.c 실행결과

```

Microsoft Visual Studio 디버그 콘솔
10
    0 0
    1 1
    2 3
10
    20 100
  
```

- 전역변수

:함수 외부에서 선언되는 변수
외부변수라고도 부른다.

일반적으로 프로젝트의 모든 함수나 블록에서 참조할 수 있다.

선언되면 자동으로 초기값이 자료형에 맞는 0으로 지정된다.

함수나 블록에서 전역변수와 같은 이름으로 지역변수를 선언할 수 있다.

프로젝트의 다른 파일에서도 참조가 가능

* 예제코드 globalvar.c

```

#include <stdio.h>

double getArea(double);
double getCircum(double);

double PI = 3.14;
int gi;

int main(void) {
    double r = 5.87;
    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);
    printf("gi: %d\n", gi);

    return 0;
}

double getArea(double r) {
    return r * r * PI;
}
  
```

* 예제코드 circumference.c

```

extern double PI;

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

* 실행결과

```

Microsoft Visual Studio 디버그 콘솔
면적: 108.19
둘레1: 36.88
둘레2: 36.86
PI: 3.141592
gi: 0
  
```

- 전역변수와 지역변수를 이용한 피보나츠 수의 출력 예제코드

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

int count;

void fibonacci(int prev_number, int number);

void main() {
    auto prev_number = 0, number = 1;

    printf("피보나치를 몇 개 구할까요?(3 이상) >> ");

    scanf("%d", &count);
    if (count <= 2)
        return 0;

    printf("1 ");
    fibonacci(prev_number, number);
    printf("\n");
}

void fibonacci(int prev_number, int number) {
    static int i = 1;

    while (i++ < count) {
        int next_num = prev_number + number;
        prev_number = number;
        number = next_num;
        printf("%d ", next_num);
        fibonacci(prev_number, number);
    }
}
```

<실행결과>

Microsoft Visual Studio 디버그 콘솔

```
피보나치를 몇 개 구할까요?(3 이상) >> 20
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
```

3.2 정적 변수와 레지스터 변수

- **auto, register, static, extern**
: 기억부류 auto와 register는 지역변수에만 이용이 가능하고 static은 지역과 전역

모든 변수에 이용 가능하다. 그리고 extern은 전역변수에만 사용이 가능하다.

기억부류 종류	전역	지역
auto	x	o
register	x	o
static	o	o
extern	o	x

- 키워드 **register**

- 레지스터 변수는 변수의 저장공간이 일반 메모리가 아니라 CPU 내부의 레지스터에 할당되는 변수
- 레지스터 변수는 키워드 register를 자료형 앞에 넣어 선언한다.
- 레지스터 변수는 일반 메모리에 할당되는 변수가 아니므로 주소연산자 &를 사용할 수 없다.
- 주로 레지스터 변수는 처리 속도를 증가시키려는 변수에 이용한다.

- 예제코드 registervar.c

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

int main(void) {
    register int sum = 0;

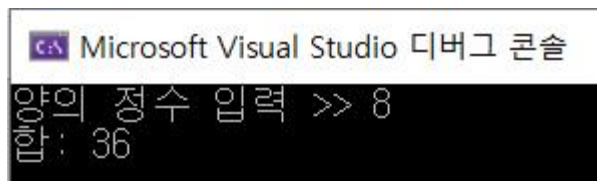
    int max;
    printf("양의 정수 입력 >> ");
    scanf("%d", &max);

    for (register int count = 1; count <= max; count++)
        sum += count;

    printf("합: %d\n", sum);

    return 0;
}
```

<실행결과>



```
Microsoft Visual Studio 디버그 콘솔
양의 정수 입력 >> 8
합: 36
```

- 키워드 **static**

- 변수 선언에서 자료형 앞에 키워드 **static**을 넣어 정적변수를 선언할 수 있다.
- 정적변수는 초기 생성된 이후 메모리에서 제거되지 않으므로 지속적으로 저장 값을 유지하거나 수정할 수 있는 특성이 있다.
- 정적변수는 초기값을 지정하지 않으면 자동으로 자료형에 따라 0이나 'w0' 또는 NULL 값이 저장된다.
- 정적변수의 초기화는 단 한번만 수행된다.

- 정적 지역변수

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

- 예제코드 staticlocal.c

```
#include <stdio.h>

void increment(void);

int main(void) {
    for (int count = 0; count < 3; count++) {
        increment();
    }
}

void increment(void) {
    static int sindex = 1;
    auto int aindex = 1;

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

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

- 정적 전역변수

- 함수 외부에서 정적으로 선언되는 변수
- 정적 전역변수는 선언된 파일 내부에서만 참조가 가능한 변수이다.
- 프로그램이 크고 복잡하면 원하지 않는 전역변수의 수정과 같은 부작용의 위험성이 항상 존재한다.

- 정적 전역변수 선언과 사용예제

* stativgvar.c

```
#include <stdio.h>

static int svar;
int gvar;

void increment();
void testglobal();

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

    testglobal();
    printf("전역 변수: %d\n", gvar);
}

void increment() {
    svar++;
}
```

* gfunc.c

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

void testglobal() {
    extern gvar;
    gvar = 10;
}
```

<실행결과>

Microsoft Visual Studio 디버그 콘솔

```
함수 increment() 가 총 5번 호출되었습니다.
전역 변수: 10
```

- 지역변수와 정적변수의 사용예제

```
#include <stdio.h>

void process();

int main() {
    process();
    process();
    process();

    return 0;
}

void process() {
    static int sx;
    int x = 1;

    printf("%d %d\n", x, sx);

    x += 3;
    sx += x + 3;
}
```

Microsoft Visual Studio 디버그 콘솔

```
1 0
1 7
1 14
```


3.3 메모리 영역과 변수 이용

- 메모리 영역

: 이러한 메모리 영역은 변수의 유효범위와 생존기간에 결정적 역할을 하며, 변수는 기억부류에 따라 할당되는 메모리 공간이 달라진다.
메인 메모리의 영역은 프로그램 실행 과정에서 데이터(data) 영역, 힙(heap) 영역, 스택(stack) 영역 세 부분으로 나뉜다.

• 데이터(data) 영역

- 전역변수와 정적변수가 할당되는 저장공간
- 메모리 주소가 낮은 값에서 높은 값으로 저장 장소가 할당
- 프로그램이 시작되는 시점에 정해진 크기대로 고정된 메모리 영역이 확보

• 힙(heap) 영역

- 동적 할당되는 변수가 할당되는 저장공간
- 프로그램이 실행되면서 영역크기가 계속적으로 변한다
- 메모리 주소가 낮은 값에서 높은 값으로 사용하지 않는 공간이 동적으로 할당

• 스택(stack) 영역

- 함수 호출에 의한 형식 매개변수 그리고 함수 내부의 지역변수가 할당되는 저장공간
- 메모리 주소가 높은 값에서 낮은 값으로 저장 장소가 할당

- 변수의 이용

: 일반적으로 전역변수의 사용을 자제하고 지역변수를 주로 이용한다.
실행 속도를 개선하고자 하는 경우에 제한적으로 레지스터 변수를 이용
해당 파일 내부에서만 변수를 공유하고자 하는 경우 정적 전역변수를 이용
함수나 블록 내부에서 함수나 블록이 종료되더라도 계속적으로 값을 저장하고 싶을 때는 정적 지역변수를 이용

• 변수의 종류

선언위치	상세 종류	키워드		유효범위	기억장소	생존시간
전역	전역 변수	참조선언	extern	프로그램 전역	메모리 (데이터 영역)	프로그램 실행시간
	정적 전역변수	static		파일내부		
지역	정적 지역변수	static		함수나 블록 내부	레지스터	함수 또는 블록 실행시간
	레지스터 변수	register			메모리 (스택 영역)	
	자동 지역변수	auto (생략가능)				

• 변수의 유효 범위

구분	종류	메모리할당 시기	동일 파일 외부 함수에서의 이용	다른 파일 외부 함수에서의 이용	메모리제거 시기
전역	전역변수	프로그램 시작	o	o	프로그램 종료
	정적 전역변수	프로그램 시작	o	x	프로그램 종료
지역	정적 지역변수	프로그램 시작	x	x	프로그램 종료
	레지스터 변수	함수(블록) 시작	x	x	함수(블록) 종료
	자동 지역변수	함수(블록) 시작	x	x	함수(블록) 종료

• 변수의 초기값

지역, 전역	종류	자동 저장되는 기본 초기값	초기값 저장
전역	전역변수	자료형에 따라 0이나 'W0' 또는 NULL 값이 저장됨	프로그램 시작 시
	정적 전역변수		
지역	정적 지역변수	쓰레기값이 저장됨	함수나 블록이 실행될 때마다
	레지스터 변수		
	자동 지역변수		

• 전역변수와 지역변수의 선언과 참조 예제코드

* storageclass.c

```
#include <stdio.h>

void infunction(void);
void outfunction(void);

int global = 10;
static int sglobal = 20;

int main(void) {
    auto int x = 100;

    printf("%d %d %d\n", global, sglobal, x);
    infunction(); outfunction();
    infunction(); outfunction();
    infunction(); outfunction();
    printf("%d %d %d\n", global, sglobal, x);

    return 0;
}

void infunction(void) {
    auto int fa = 1;
    static int fs;

    printf("%d %d %d %d\n", ++global, ++sglobal, fa, ++fs);
}
```

* out.c

```
#include <stdio.h>

void outfunction() {
    extern int global, sglobal;

    printf("%d %d %d\n", ++global);
}
```

Microsoft Visual Studio 디버깅 콘솔

```
10 20 100
11, 21, 1, 1
12
13, 22, 1, 2
14
15, 23, 1, 3
16
16 23 100
```

- 전역변수와 지역변수의 선언과 참조를 이용하여 은행 계좌의 입출금 구현 예제코드

```
#include <stdio.h>

int total = 10000;

void save(int);
void withdraw(int);

int main(void) {
    printf(" 입금액   출금액   총입금액   총출금액   잔고\n");
    printf("=====");
    printf("%4d\n", total);
    save(50000);
    withdraw(30000);
    save(60000);
    withdraw(20000);
    printf("=====");

    return 0;
}

void save(int money) {
    static int amount;
    total += money;
    amount += money;
    printf("%7d %17d %20d\n", money, amount, total);
}

void withdraw(int money) {
    static int amount;
    total -= money;
    amount += money;
    printf("%15d %20d %9d\n", money, amount, total);
}
```

Microsoft Visual Studio 디버그 콘솔

입금액	출금액	총입금액	총출금액	잔고
				10000
50000		50000		60000
	30000		30000	30000
60000		110000		90000
	20000		50000	70000

4. 13장 구조체와 공용체

4.1 구조체와 공용체

- 구조체 개념과 정의

- 구조체 개념

- : 정수나 문자, 실수나 포인터 그리고 이들의 배열 등을 묶어 하나의 자료형으로 이용하는 것
- : 연관성이 있는 다른 개별적인 자료형의 변수들을 하나의 단위로 묶은 새로운 자료형
- : 새로이 만들어진 자료형을 유도 자료형이라 한다.

- 구조체 정의

- : 변수의 선언과는 다른 것으로 변수선언에서 이용될 새로운 구조체 자료형을 정의하는 구문
- : 구조체를 사용하려면 먼저 구조체를 만들 구조체 틀을 정의하여야 한다.
- : 구조체를 정의하는 방법은 키워드 struct 다음에 구조체 태그이름을 기술하고 중괄호를 이용하여 원하는 멤버를 여러 개의 변수로 선언하는 구조이다.

- 구조체 변수 선언과 초기화

- 구조체 변수 선언

- : struct 구조체태그이름 변수형; 으로 선언한다.
ex) struct account yours;

: 구조체 정의와 변수 선언을 함께하는 문장

```
ex) struct account
{
    char name[12];
    int actnum;
    double balance;
} myaccount;
struct account youraccount;
```

: 구조체 태그이름이 없는 변수 선언 방법

```
ex) struct
{
    char name[12];
    int actnum;
    double balance;
} youraccount;
```

- 구조체 변수 초기화

- : 중괄호 내부에서 구조체의 각 멤버 정의 순서대로 초기값을 쉼표로 구분하여 기술
- : 배열과 같이 초기값에 기술되지 않은 멤버값은 자료형에 따라 기본값인 0, 0.0, 'W0' 등으로 저장된다.
- : struct 구조체태그이름 변수명={초기값1, 초기값2, 초기값3, ...};

- * 예제코드 structbasic.c

구조체 정의와 구조체 변수 선언 예제

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

struct account {
    char name[12];
    int actnum;
    double balance;
};

int main(void) {
    struct account mine = { "홍길동", 1001, 300000 };
    struct account yours;

    strcpy(yours.name, "이동원");

    yours.actnum = 1002;
    yours.balance = 500000;

    printf("구조체크기: %d\n", sizeof(mine));
    printf("%s %d %.2f\n", mine.name, mine.actnum, mine.balance);
    printf("%s %d %.2f\n", yours.name, yours.actnum, yours.balance);

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
구조체크기: 24
홍길동 1001 300000.00
이동원 1002 500000.00
```

- 구조체 활용

• 구조체 변수의 대입과 동등비교

: 구조체 멤버마다 모두 대입할 필요 없이 변수 대입으로 한번에 모든 멤버의 대입이 가능하다.

ex) `struct student hong = {...};`
`struct student one;`
`one = hong;`

: `(one == hong)`와 같은 동등 비교는 사용할 수 없다.

* 구조체의 동등비교 예제코드 structstudent.c

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

int main(void) {
    struct student
    {
        int snum;
        char* dept;
        char name[12];
    };
    struct student hong = { 201800001, "컴퓨터정보공학과", "홍길동" };
    struct student na = { 201800002 };
    struct student bae = { 201800003 };

    scanf("%s", na.name);

    na.dept = "컴퓨터정보공학과";
    bae.dept = "기계공학과";
    memcpy(bae.name, "배상문", 7);
    strcpy(bae.name, "배상문");
    strcpy_s(bae.name, 7, "배상문");

    printf("[%d, %s, %s]\n", hong.snum, hong.dept, hong.name);
    printf("[%d, %s, %s]\n", na.snum, na.dept, na.name);
    printf("[%d, %s, %s]\n", bae.snum, bae.dept, bae.name);

    struct student one;
    one = bae;
    if (one.snum == bae.snum)
        printf("학번이 %d으로 동일합니다.\n", one.snum);

    if (one.snum == bae.snum && !strcmp(one.name, bae.name) && !strcmp(one.dept, bae.dept))
        printf("내용이 같은 구조체입니다.\n");

    return 0;
}
```

C# Microsoft Visual Studio 디버그 콘솔

```
나한국
[201800001, 컴퓨터정보공학과, 홍길동]
[201800002, 컴퓨터정보공학과, 나한국]
[201800003, 기계공학과, 배상문]
학번이 201800003으로 동일합니다.
내용이 같은 구조체입니다.
```

- char 포인터와 char 배열의 비교

char 포인터	char 배열
char *dept	char name[12];
char *dept = "컴퓨터정보공학과";	char name[12] = "나한국";
변수 dept는 포인터로 단순히 문자열 상수를 다루는 경우 효과적	변수 name은 배열로 12바이트 공간을 가지며 문자열을 저장하고 수정 등이 필요한 경우 효과적
dept = "컴퓨터정보공학과";	name = "나한국"; //오류
단지 문자열 상수의 첫 주소를 저장하므로 문자열 자체를 저장하거나 수정하는 것은 불가능하므로 다음 구문은 사용 불가능	문자열 자체를 저장하는 배열이므로 문자열의 저장 및 수정이 가능하고 문자열 자체를 저장하는 다음 구문 사용도 가능
strcpy(dept, "컴퓨터정보공학과");	strcpy(name, "배상문");
scanf("%s",dept);	scanf("%s", name);

- 공용체 활용

- 공용체 개념

- : 동일한 저장 장소에 여러 자료형을 저장하는 방법
- : 공용체는 서로 다른 자료형의 값을 동일한 저장공간에 저장하는 자료형

- union을 사용한 공용체 정의 및 변수 선언

- : 공용체 변수의 크기는 멤버 중 가장 큰 자료형의 크기로 정해진다.
- : 공용체 멤버는 모든 멤버가 동일한 저장 공간을 사용하므로 동시에 여러 멤버가 값을 동시에 저장하여 이용할 수 없으며, 마지막에 저장된 단 하나의 멤버 자료값만을 저장한다.
- : 공용체의 초기화 값은 공용체 정의 시 처음 선언한 멤버의 초기값으로만 저장 가능하다.

- 공용체 멤버 접근

- : 공용체 변수로 멤버를 접근하기 위해서는 구조체와 같이 접근연산자 .를 사용

- * 공용체 정의와 변수 선언 및 사용 예제코드

```
#include <stdio.h>

union data {
    char ch;
    int cnt;
    double real;
} data1;

int main(void) {
    union data data2 = { 'A' };
    union data data3 = data2;
}
```



```

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


data1.ch = 'a';
printf("%c %d %f\n", data1.ch, data1.cnt, data1.real);

data1.cnt = 100;
printf("%c %d %f\n", data1.ch, data1.cnt, data1.real);

data1.real = 3.156759;
printf("%c %d %f\n", data1.ch, data1.cnt, data1.real);

return 0;
}

```

 Microsoft Visual Studio 디버그 콘솔

```

8 8
a 97 0.000000
d 100 0.000000
N -590162866 3.156759

```

4.2 자료형 재정의

- 자료형의 재정의 typedef

• typedef 구문

- : 이미 사용되는 자료 유형을 다른 새로운 자료형 이름으로 재정의할 수 있도록 하는 키워드
- : 일반적으로 자료형을 재정의하는 이유는 프로그램의 시스템 간 호환성과 편의성을 위해 필요하다.
- : 문장 typedef도 일반 변수와 같이 그 사용 범위를 제한한다.

* 자료형 재정의 이용 예제코드

```

#include <stdio.h>

typedef unsigned int budget;

int main(void) {
    budget year = 24500000;
    typedef int profit;
    profit month = 4600000;

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

    return 0;
}

void test(void) {
    budget year = 24500000;
}

```


Microsoft Visual Studio 디버그 콘솔

올 예산은 24500000, 이달의 이익은 4600000 입니다.

- 구조체 자료형 재정의

•struct를 생략한 새로운 자료형

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

: date가 아닌 datatype 등 다른 이름으로도 재정의가 가능

: 구조체 정의와 type를 함께 이용한 자료형의 정의

ex) typedef struct

```
{
    char title[30];
    char company[30];
    char kinds[30];
    date release;
} software;
```

* 문장 typedef를 이용하여 구조체의 자료유형을 다른 이름으로 재정의하여 이용 예제코드

```
#include <stdio.h>

struct date
{
    int year;
    int month;
    int day;
};

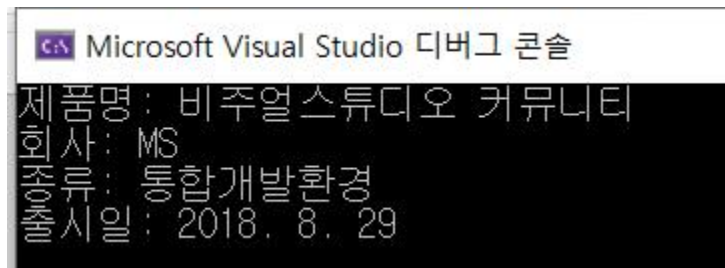
typedef struct date date;

int main(void) {
    typedef struct
    {
        char title[30];
        char company[30];
        char kinds[30];
        date release;
    } software;

    software vs = { "비주얼스튜디오 커뮤니티", "MS", "통합개발환경", {2018,8,29} };

    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;
}
```



4.3 구조체와 공용체의 포인터와 배열

- 구조체 포인터

- 포인터 변수 선언

: 포인터는 각각의 자료형 저장 공간의 주소를 저장하듯이 구조체 포인터는 구조체의 주소값을 저장할 수 있는 변수이다.

- 포인터 변수의 구조체 멤버 접근 연산자 ->

: 구조체 포인터 멤버 접근연산자 ->는 p->name과 같이 사용한다.

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

: 연산식 *p.name은 접근연산자(.)가 간접연산자(*)보다 우선순위가 빠르므로 (*p).name으로도 사용가능하다.

- 공용체 포인터

: 공용체 변수도 포인터 변수 사용이 가능하며, 공용체 포인터 변수로 멤버를 접근하려면 접근연산자 ->를 이용한다.

* 공용체 정의와 변수 선언 및 사용 예제코드

```
#include <stdio.h>

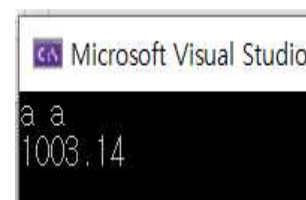
int main(void) {
    union data
    {
        char ch;
        int cnt;
        double real;
    };

    typedef union data udata;

    udata value, * p;

    p = &value;
    p->ch = 'a';
    printf("%c %c\n", p->ch, (*p).ch);
    p->cnt = 100;
    printf("%d", p->cnt);
    p->real = 3.14;
    printf("%.2f\n", p->real);

    return 0;
}
```



- 구조체 배열

• 구조체 배열 변수 선언

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

* 구조체 배열을 선언한 후 출력 처리

```
#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", "강좌구분", "학점", "시수" };

int main(void) {
    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;
}
```

Microsoft Visual Studio 디버그 콘솔

배열크기: 5

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

5. 14장 함수와 포인터 활용

5.1 함수의 인자전달 방식

- 값에 의한 호출과 참조에 의한 호출

- 함수에서 값의 전달

- : C언어는 함수의 인자 전달 방식이 기본적으로 값에 의한 호출방식이다.
- : 값에 의한 호출 방식이란 함수 호출 시 실인자의 값이 형식인자에 복사되어 저장된다는 의미이다.
- : 값에 의한 호출 방식을 사용해서는 함수 외부의 변수를 함수 내부에서 수정할 수 없는 특징이 있다.

- 함수에서 주소의 전달

- : C언어에서 포인터를 매개변수로 사용하면 함수로 전달된 실인자의 주소를 이용하여 그 변수를 참조할 수 있다.

- 배열의 전달

- 배열이름으로 전달

- : 함수의 매개변수로 배열을 전달하는 것은 배열의 첫 원소를 참조 매개변수로 전달하는 것과 동일하다.
- : 배열크기에 관계없이 배열 원소의 합을 구하는 함수를 만들려면 배열크기고 하나의 인자로 사용해야 한다.
- : 함수원형에서 매개변수는 배열이름을 생략하고 `double []`와 같이 기술할 수 있다.
- : 함수호출에서 배열 인자에는 반드시 배열이름으로 `sum(data,5)`와 같이 기술해야 한다.

- 배열크기 계산방법

- : 연산자 `sizeof`를 이용한 식 (`sizeof(배열이름) / sizeof(배열원소)`)의 결과는 배열 크기

- 다차원 배열 전달

- : 다차원 배열을 인자로 이용하는 경우, 함수원형과 함수정의의 헤더에서 첫 번째 대괄호 내부의 크기를 제외한 다른 모든 크기는 반드시 기술되어야 한다.
- : 이차원 배열의 행의 수는 다음과 같이 (`sizeof(x) / sizeof(x[0])`)로 계산할 수 있다.
- : 이차원 배열의 열의 수는 다음과 같이 (`sizeof(x[0]) / sizeof(x[0][0])`)로 계산한다.

- 가변인자

- 가변인자가 있는 함수머리

- : 출력할 인자의 수와 자료형은 인자 `_Format`에 `%d` 등으로 표현되어 있다.
- : 함수에서 인자의 수와 자료형이 결정되지 않은 함수 인자 방식을 가변 인자라 한다.

- 가변 인자가 있는 함수 구현

- : 가변 인자를 구현하려면 가변인자 선언, 가변인자 처리 시작, 가변인자 얻기, 가변인자 처리 종료 4단계가 필요
- : 헤더파일 stdarg.h

- * 함수에서 배열을 활용한 예제코드

```
#include <stdio.h>

void aryprocess(int* ary, int SIZE);

int main(void) {
    int data[] = { 1,3,5,7,9 };

    int aryLength = sizeof(data) / sizeof(int);
    aryprocess(data, aryLength);
    for (int i = 0; i < aryLength; i++)
        printf("%d ", *(data + i));
    printf("\n");

    return 0;
}

void aryprocess(int* ary, int SIZE) {
    for (int i = 0; i < SIZE; i++)
        (*ary++)++;
}
```

Microsoft Visual S

2 4 6 8 10

5.2 포인터 전달과 반환

- 매개변수와 반환으로 포인터 사용

- 주소연산자 &

- : 함수에서 매개변수를 포인터로 이용하면 결국 참조에 의한 호출이 된다

- 상수를 위한 const 사용

- 키워드 const

- : 수정을 원하지 않는 함수의 인자 앞에 키워드 const를 삽입

- 함수의 구조체 전달과 반환

- 복소수를 위한 구조체

- : 구조체 complex를 이용하여 복소수 연산에 이용되는 함수
- : 구조체 complex는 실수부와 허수부를 나타내는 real과 img를 멤버로 구성

- 인자와 반환형으로 구조체 사용

- : 함수 paircomplex1()는 인자인 복소수의 켤레 복소수를 구하여 반환하는 함수
- : 구조체는 함수의 인자와 반환값으로 이용이 가능

- * 책 정보를 표현하는 구조체 전달 예제코드

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

typedef struct book
{
    char title[50];
    char author[50];
    int ISBN;
} book;

void print(book* b);

int main()
{
    book python = { "파이썬웹프로그래밍", "김석훈", 2398765 };
    book java;
    strcpy(java.title, "절대자바");
    strcpy(java.author, "강환수");
    java.ISBN = 123987;
    print(&java);
    print(&python);

    return 0;
}

void print(book* b)
{
    printf("제목: %s, ", b->title);
    printf("저자: %s, ", b->author);
    printf("ISBN: %d\n", b->ISBN);
}
```

 Microsoft Visual Studio 디버깅 콘솔

```
제목: 절대자바, 저자: 강환수, ISBN: 123987
제목: 파이썬웹프로그래밍, 저자: 김석훈, ISBN: 2398765
```

5.3 함수 포인터와 void 포인터

- 함수 포인터

- 함수 주소 저장 변수

- : 포인터의 장점은 다른 변수를 참조하여 읽거나 쓰는 것이 가능하다는 것이다.

- : 함수 포인터는 함수의 주소값을 저장하는 포인터 변수이다.

- 함수 포인터 배열

- 함수 포인터 배열 개념

- : 함수 포인터 배열은 함수 포인터가 원소인 배열이다.

- 함수 주소 저장 선언

- : 반환자료형 (*배열이름[배열크기])(자료형1 매개변수이름1, ...); or
반환자료형 (*배열이름[배열크기])(자료형1,...);

- void 포인터

- void 포인터 개념

- : 자료형을 무시하고 주소값만을 다루는 포인터

- : void 포인터에는 일반 변수 포인터는 물론 배열과 구조체 심지어 함수 주소도 담을 수 있다.

- void 포인터 활용

- : 모든 주소를 저장할 수 있지만 가리키는 변수를 참조하거나 수정이 불가능

- : 변수를 참조하기 위해서는 자료형 반환이 필요

* 함수 포인터 배열의 활용 예제코드

```
#include <stdio.h>

int add(int a, int b);
int mult(int a, int b);
int subtr(int a, int b);

int main(void) {
    int(*pfunary[3])(int, int);
    pfunary[0] = add;
    pfunary[1] = mult;
    pfunary[2] = subtr;
}
```



```

char* ops = "+-";
char op;
while (op = *ops++)
    switch (op) {
        case '+': printf("%c 결과: %d\n", op, pfunary[0](3, 5));
                    break;
        case '-': printf("%c 결과: %d\n", op, pfunary[2](3, 5));
                    break;
        case '*': printf("%c 결과: %d\n", op, pfunary[1](3, 5));
                    break;
    }
return 0;
}

int add(int a, int b) {
    return a + b;
}

int mult(int a, int b) {
    return a * b;
}

int subt(int a, int b) {
    return a - b;
}

```

Microsoft Visual Studio

```

* 결과: 15
+ 결과: 8
- 결과: -2

```

6. 15장 파일 처리

6.1 파일 기초

- 텍스트 파일과 이진 파일

- 텍스트 파일, 이진파일

- : 파일은 텍스트 파일과 이진 파일 두 가지 유형으로 나뉜다.
- : 텍스트 파일은 문자 기반의 파일로서 내용이 아스키 코드와 같은 문자 코드값으로 저장된다.
- : 이진 파일은 텍스트 파일과 다르게 그림 파일, 동영상 파일, 실행 파일과 같이 각각의 목적에 알맞은 자료가 이진 형태로 저장되는 파일.
- : 이진 파일은 컴퓨터 내부 형식으로 저장되는 파일
- : 이진 파일에서 자료는 메모리 자료 내용에서 어떤 변환도 거치지 않고 그대로 파일에 기록된다. 입출력 속도 또한 텍스트 파일에 비해 빠르다.

- 입출력 스트림

- : 자료의 입력과 출력은 자료의 이동, 자료가 이동하려면 이동 경로가 필요
- : 입출력 시 이동 통로가 바로 **입출력스트림** 이다.
- : 입력 스트림 = 다른 곳에서 프로그램으로 들어오는 경로
- : 출력 스트림 = 프로그램에서 다른 곳으로 나가는 경로

- 파일 스트림 이해

- : 프로그램에서 보조기억장치에 파일로 정보를 저장하거나 파일에서 정보를 참조하려면 파일에 대한 파일 스트림을 먼저 연결해야 한다.
- : 파일 스트림이란 보조기억장치의 파일과 프로그램을 연결하는 전송경로

- 파일스트림 열기

- fopen()으로 파일 스트림 열기

- : 프로그램에서 특정한 파일과 파일 스트림을 연결하기 위해서는 함수 fopen() 또는 fopen_s()를 이용한다.
- : FILE은 헤더 파일 stdio.h에 정의되어 있는 구조체 유형
- : 함수 fopen()은 인자가 파일이름과 파일열기 모드이며, 파일 스트림 연결에 성공하면 파일 포인터를 반환하며, 실패하면 NULL을 반환한다.
- : fopen_s()는 파일 스트림 연결에 성공하면 정수0을 반환 실패하면 양수를 반환

- fclose()로 파일 스트림 닫기

- : fclose()는 fopen()으로 연결한 파일 스트림을 닫는 기능을 수행한다.

6.2 텍스트 파일 입출력

- 파일 문자열 입출력

- fgetc()와 fputc()

- : fgets()는 파일로부터 한 행의 문자열을 입력받는 함수
- : fgets()는 파일로부터 문자열을 개행문자(wn)까지 읽어 마지막 개행문자를 'w0' 문자로 바꾸어 입력 버퍼 문자열에 저장
- : fputs()는 파일로 한행의 문자열을 출력하는 함수 , 문자열을 한 행에 출력

- **feof()와 ferror()**

- : feof()은 파일 스트림의 EOF표시를 검사하는 함수
- : ferror()는 파일 처리에서 오류가 발생했는지 검사하는 함수

- **파일 문자 입출력**

- **fgetc()와 fputc()**

- : fgetc()와 getc()는 파일로부터 문자 하나를 입력받는 함수
- : fputc()와 putc()는 문자 하나를 파일로 출력하는 함수

6.3 이진 파일 입출력 , 파일 접근 처리

- **텍스트와 이진 파일 입력과 출력**

- **fprintf()와 fsanf_s()**

- : fprintf()와 fsanf(), fsanf_s()는 자료의 입출력을 텍스트 모드로 처리

- **fwrite()와 fread()**

- : fwrite()는 바이트 단위로 원하는 블록을 파일에 출력하기 위한 함수
- : 이진 모드로 블록 단위 입출력을 처리하려면 함수 fwrite()와 fread()를 이용

- **순차 접근과 임의 접근**

- **파일 위치**

- : 파일 위치는 파일 내부를 바이트 단위로 파일 내부 위치를 나타내는 값
- : 파일 지시자 또는 파일 표시자 라고도 부른다.
- : 파일 마지막에는 파일의 마지막임을 알리는 EOF표시가 있다.

- **파일 스트림 연결 시 파일 위치**

- : 파일을 처음으로 열면 모드에 관계없이 파일 위치는 모두 0

- **파일 순차적 접근과 임의 접근**

- : 파일 위치를 처음부터 하나씩 증가시키면서 파일을 참조하는 방식을 순차적 접근이라 한다.
- : 순차적 접근과는 다르게 파일의 어느 위치든 바로 참조하는 방식을 임의 접근이라 한다.

- **파일의 임의 접근 함수**

- **fseek()**

- : 파일의 임의 접근을 처리하기 위해 파일 위치를 자유로 이동하는 fseek()가 필요

: fseek()에서 세 번째 인자는 오프셋을 계산하는 기준으로 정수형 기호 상수로 다음 세가지 중의 하나를 이용할 수 있다.

기호	값	의미
SEEK_SET	0	파일의 시작 위치
SEEK_CUR	1	파일의 현재 위치
SEEK_END	2	파일의 끝 위치

- **파일 열기 다양한 모드**

- : 파일열기 종류에는 텍스트 파일인 경우 "r","w","a","r+","w+","a+" 등의 종류가 있다.
- : 읽기모드 r은 읽기가 가능한 모드, 쓰기는 불가능
- : 쓰기모드 w는 파일 어딘든 쓰기가 가능한 모드, 읽기는 불가능
- : 추가모드 a는 파일 중간에 쓸 수 없으며 파일 마지막에 추가적으로 쓰는 것만 가능한 모드, 읽기는 불가능

7. 맺음말

포트폴리오를 시작하기 전 저는 어떤 방식으로 만들어야 하여야 하는지 생각을 해 보았습니다. 지금까지 C언어를 배우면서 연습해본 코드들을 모으지 않았고 제대로 머릿속에 들어오지 않았다고 느껴져 C언어의 개념, 특징들을 정리해보고 11장부터 15장까지의 내용을 다시 복습하면서 내용을 정리하고 예제코드들을 살펴보면서 진행을 해보며 느낀 점으로 2가지가 있습니다.

첫 번째로 11장부터 15장의 내용을 정리하면서 13장 구조체와 공용체, 15장의 파일 처리에 대한 내용 이해가 부족하다고 느껴졌으며 다른 단원의 경우 이해했다고 생각했던 것도 다시 확인해보니 부족했던 부분이 많은 것을 보고 복습이 중요한 것을 다시 한번 느끼게 되었습니다.

두 번째로는 지금까지 C언어를 학습하면서 진행한 자료들을 모아두지 않으면 나중에 나의 능력과 실력을 증명할 수 있는 자료가 없을 것이라고 느끼고 C언어를 배우며 습득해온 내용을 자료화하고 연습했거나 진행한 작품에 대한 자료들을 모아놔야 겠다고 느끼게 되었습니다.

포트폴리오 만들고 난 후 몰랐던 부분을 다시 확인하는 계기가 되었고 미래에 대한 계획이 생기게 되었습니다. 미래에 나의 실력과 능력을 증명할 수 있는 자료들을 열심히 모으고 만들어서 내가 하고 싶었던 일을 할 수 있게 열심히 준비하려 합니다.