

C 언어 Portfolio

학 과 : 컴퓨터정보공학과
학 번 : 20193418
성 명 : 김현준

목차

1. 강의 계획서

A. 교재 소재

2. 1주차 수업

A. 문자와 문자열

- a. 문자와 문자열 개념
- b. 문자열 관련 함수
- c. 과제

3. 2주차 수업

A. 문자와 문자열

- a. 문자열 관련 함수
- c. 여러 문자열 처리

B. 변수 유효범위

- a. 전역변수와 지역변수
- b. 정적 변수와 레지스터 변수
- c. 과제

4. 3주차 수업

A. 변수 유효범위

- a. 메모리 영역과 변수 이용
- b. 과제

5. 4주차 수업

A. 구조체와 공용체

- a. 구조체와 공용체 개념
- b. 자료형 재정의
- c. 과제

6. 5주차 수업

A. 구조체와 공용체

- a. 구조체와 공용체의 포인터 와 배열

B. 함수와 포인터 활용

- a. 함수의 인자전달 방식
- b. 포인터 전달과 반환

7. 6주차 수업

A. 함수와 포인터 활용

- a. 포인터 전달과 반환
- b. 함수 포인터void 포인터

B. 파일처리

- a.파일 기초

8. 7주차 수업

A. 파일처리

- a.파일 기초
- b.텍스트 파일 입출력
- c.이진파일 입출력

강의계획서



아시아 직업교육 허브대학

2020 학년도 1 학기	전공	컴퓨터정보공학과(위)	학부	컴퓨터공학부
과 목 명	C애플리케이션구현(2016003-PH)			

강의실 과 강의시간	월:11(3-217),12(3-217),13(3-217),14(3-217)	학점	4
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	인피니티박스	
-----	-----------	---------------	--------	--

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
--------	------------------------------------

강의계획서



아시아 직업교육 허브대학

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

강의계획서



아시아 직업교육 허브대학

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 포인터활용
미리읽어오기	교재 14장
과제,시험,기타	수업 중에 제시함

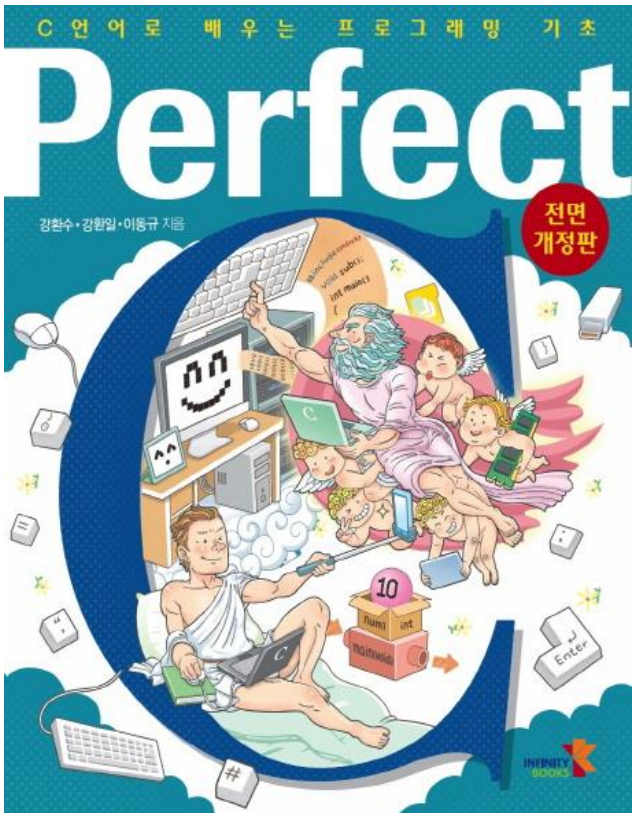
강의계획서



아시아 직업교육 허브대학

13 주차	[13주]
학습주제	파일처리
목표및 내용	15장 파일처리
미리읽어오기	교재 15장
과제,시험,기타	수업 중에 제시함
14 주차	[14주]
학습주제	항상심화강좌(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>

교재 소개



Perfect C

C언어로 배우는 프로그래밍 기초

강환수, 강환일, 이동규 지음

2016년 01월11일 출간

인피니티북스

1주차 수업 문자와 문자열

문자와 문자열의 개념

• 문자

- 영어의 알파벳이나 한글의 한 글자를 작은 따옴표로 둘러싸서 'A'와 같이 표기한다.

* C언어에서 저장공간 크기 1바이트인 자료형 char로 지원

- 작은 따옴표에 의해 표기된 문자를 문자 상수라 한다.

• 문자열 (string)

- 문자의 모임인 일련의 문자

* 일련의 문자 앞 뒤로 큰 따옴표로 둘러싸서 "java"로 표기

- 큰 따옴표에 의해 표기된 문자열을 문자열 상수라 한다.

* "A"처럼 문자 하나도 큰 따옴표로 둘러싸서 문자열 상수

* 'ABC'처럼 작은 따옴표로 둘러싸도 문자가 될 수 없으며 오류가 발생한다.

문자와 문자열의 선언

• char 형 변수에 문자를 저장

- 문자열을 저장하기 위한 자료형을 따로 제공하지 않는다.

- 문자 배열을 선언하여 각각의 원소에 문자를 저장 한다.

- 문자열의 마지막을 의미하는 NULL문자 '\0'가 마지막에 저장된다.

- 문자열이 저장되는 배열의 크기

* 반드시 저장될 문자 수보다 1이 커야 널(NULL) 문자를 문자열의 마지막으로 인식

* 문자열의 마지막에 널(NULL) 문자가 없다면 출력과 같은 문자열 처리에 문제가 발생한다.

예) 배열 csharp의 크기를 3으로 선언한 후 배열 csharp에 문자열 "C#"을 저장, 마지막 원소인 csharp[2]에 '\0'을 저장한다.

```
char ch = 'A';

char csharp[3];
csharp[0] = 'C'; csharp[1] = '#'; csharp[2] = '\0';
```

•배열 선언 시 초기화 방법

- 중괄호를 사용한다.
- 문자 하나 하나를 쉼표로 구분하여 입력하고 마지막 문자로 널(NULL)인 '\0' 을 삽입 한다.

예)

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

문자열을 선언하는 편리한 다른 방법

•배열 선언 시 저장할 큰 따옴표를 사용해 문자열 상수를 바로 대입

- 배열 초기화 시 배열크기는 지정하지 않는 것이 더 편리하다.
 - * 지정한다면 마지막 문자인 '\0' 을 고려해 실제 문자 수보다 1이 더 크게 배열 크기를 지정한다.
 - * 지정한 배열크기가 (문자수+1)보다 크면 나머지 부분은 모두 '\0' 문자로 채워진다.
- 만일 배열크기가 작을경우
 - * 문자열 상수가 아닌 단순한 문자 배열이 되므로 문자열 출력 등에서 문제가 발생한다

예)

```
char c[] = "C language"; //크기를 생략하는 것이 간편하다.
char c[11] = "C language"; //크기 지정 시 (문자수+1)
char go[5] = "go"; //크기가 (문자+1)보다 크면 나머지는 '\0'으로채워진다.
```

문자열 구성하는 문자 참조

•문자열을 처리하는 다른 방법

- 문자열 상수를 문자 포인터에 저장하는 방식
- 문자 포인터 변수에 문자열 상수를 저장하는 방식
- 문자열 출력도 함수 printf()에서 포인터 변수와 형식제어문자 %s
- 문자 포인터에 의한 선언으로는 문자 하나 하나의 수정은 불가능하다.

```
int i = 0;
char *java = "java";
printf("%s ", java);
```

```
while (java[i] != '\0')
    printf("%c", java[i++]);
print("\n");
```


문자열 관련 함수

• 함수 `getchar()`

- 문자의 입력에 사용된다.
- 라인 버퍼링(line buffering) 방식을 사용한다.
 - * 문자 하나를 입력해도 반응을 보이지 않다가 [enter] 키를 누르면 이전에 입력한 문자마다 입력이 실행된다.
 - * 입력한 문자는 임시 저장소인 버퍼(buffer)에 저장되었다가 [enter] 키를 만나면 함수는 버퍼에서 문자를 읽기 시작한다.
- 즉각적인 입력을 요구하는 시스템에서는 사용이 불가능 하다.

• 함수 `getche()`

- `getche()`는 버퍼를 사용하지 않는다.
 - * 문자 하나를 입력하면 바로 함수 `getche()`가 실행된다.
- 함수 `getche()`에서 입력된 문자는 바로 모니터에 표시된다.
- 함수를 이용하려면 헤더 파일 `conio.h`를 삽입한다.
- 입력 문자가 'q' 가 아니면 함수 `putchar()`에 의하여 문자가 바로 출력된다.
 - * 함수 `getche()`에 의하여 입력된 문자도 보이고 바로 `putchar()`에 의하여 출력된다.

• 함수 `getch()`

- 문자 입력을 위함 함수이다.
- 입력한 문자가 화면에 보이지 않는 특성이 있다.
- 입력된 문자를 출력함수로 따로 출력하지 않으면 입력 문자가 화면에 보이지(echo) 않는다.
- `conio.h`를 삽입해야 한다.

• 함수 `gets()` : 한 행의 문자열 입력

- 헤더파일 `stdio.h` 를 삽입한다.
- 함수 `gets()`는 [enter]키를 누를 때까지 한 행을 버퍼에 저장 한 후 입력처리 한다.
 - * 마지막에 입력된 '\n' 가 '\0' 로 교체되어 인자인 배열에 저장
 - * 한 행을 하나의 문자열로 간주하고 프로그래밍할 수 있도록 한다.

• 함수 puts(): 한 행에 문자열을 출력

- 헤더파일 stdio.h 를 삽입한다.
- 오류가 발생하면 EOF를 반환한다.
 - * 기호 상수 EOF(End Of File)
 - * 파일의 끝이라는 의미로 stdio.h 헤더파일에 정수 -1로 정의한다.
 - define EOF (-1)
- 함수 gets()와 반대로 문자열의 마지막에 저장된 '\0' 를 '\n' 로 교체하여 버퍼에 전송한다.
- 버퍼의 내용이 모니터에 출력되면 문자열이 한 행에 출력된다.

• 다양한 문자열 라이브러리 함수

- 헤더 하일 string.h에 함수원형으로 선언된 라이브러리 함수로 제공 된다.
- 문자열 비교와 복사, 그리고 문자열 연결 등과 같은 다양한 문자열 처리.
- 문자의 배열 관련 함수
 - * 자료형 siz_t 비부호 정수형(unsigned int type)
 - * 자료형 void* 아직 정해지지 않은 다양한 포인터를 의미한다.

• 함수 strcmp()

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

두 인자인 문자열에서 같은 위치의 문자를 앞에서부터 다를 때까지 비교하여 같으면 0을 반환하고, 앞이 크면 양수를 , 뒤가 크면 음수를 반환한다.

```
int strcmp(const char * s1, const char * s2);
```

두 인자 문자열을 같은 위치의 문자를 앞에서부터 다를 때까지 비교 하나 최대 n까지만 비교하여 같으면 0을 반환하고, 앞이 크면 양수를, 뒤가 크면 음수를 반환한다.

```
int strncmp(const char * s1, const char * s2, size_t maxn);
```

• 함수 strcpy

- 문자열을 복사하는 함수다.
- 함수 strcpy()는 앞 인자 문자열 dest에 뒤 인자 문자열 source를 복사한다.
- 첫 번째 인자인 dest는 복사 결과가 저장될 수 있도록 충분한 공간을 확보한다.
- 함수 strncpy()는 복사되는 최대 문자 수를 마지막 인자 maxn으로 지정하는 함수이다.

앞 문자열 dest에 처음에 뒤 문자열 null 문자를 포함한 source를 복사하여 그 복사된 문자열을 반환한다.

앞 문자열은 수정되지만 뒤 문자열은 수정될 수 없다.

```
char * strcpy(char * dest, const char * source);
```

앞 문자열 dest에 처음에 뒤 문자열 source에서 n개 문자를 복사하여 그 복사된 문자열을 반환한다.

만일 지정된 maxn이 source의 길이보다 길면 나머지는 모두 널 문자가 복사된다. 앞 문자열은 수정되지만 뒤 문자열은 수정될 수 없다.

```
char * strcpy(char * dest, const char * source, size_t maxn);
```

두번째 인자인 sizedest는 정수형으로 dest의 크기를 입력한다.

반환형 errno_t는 정수형이며 반환값은 오류번호로 성공하면 0을 반환한다. Visual C++에서는 앞으로 strcpy_s()와 strncpy_s()의 사용을 권장한다.

```
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);
```

1주차 과제

- 실습예제 1, 3, 4, 5, 6 프로그래밍 구현 , 코딩과 결과 작성

• 실습예제 1. chararray.c

-함수 printf() 를 사용한 문자와 문자열 출력

-함수 printf()

- * 형식제어문자 %c로 문자를 출력한다.

- * 배열이름 또는 문자 포인터를 사용하여 형식 제어문자%s 로 문자열을 출력한다.

-함수 pust(csharp)

- * 한줄에 문자열을 출력한 후 다음 줄에서 출력을 준비한다.

-함수 printf(c)

- *배열이름을 인자로 사용해도 문자열을 출력한다.

```
#include <stdio.h>

int main(void)
{
    //문자 선언과 출력
    char ch = 'A';
    printf("%c %d\n", ch, ch);

    //문자열 선언 방법1
    char java[] = { 'J', 'A', 'V', 'A', '\0' };
    printf("%s\n", java);

    //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;
}
```

결과.

Microsoft Visual Studio 디버그 콘솔

```
A 65
JAVA
C language
C#
C#
```

• 실습예제 3. string.c

‘\0’ 문자에 의한 문자열 분리

-문자배열로 문자열을 처리한다.

-함수 printf()

- * %s는 문자 포인터가 가리키는 위치에서 NULL 문자까지를 하나의 문자열로 인식한다.

-배열c[]


- * 처음에 문자열 “C C++ JAVA” 가 저장되고 마지막에 NULL 문자가 저장된다

```
#include <stdio.h>

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

    //문자 배열의 각 원소를 하나 하나 출력하는 방법
    c[5] = ' '; // 널 문자를 빈 문자로 바꾸어 문자열 복원
    char* p = c;
    while (*p)
        printf("%c", *p++); // (*p != '\0')도 가능하다
    printf("\n");
    return 0;
}
```

결과.

 Microsoft Visual Studio 디버그 콘솔

```
C C++ Java
C C++
Java
C C++ Java
```

• 실습예제 4. getche.c

-세 개의 while문의 입력으로 각각 “getchar()q” , “getche()q” , “getch()q” 를 입력한다.

```
#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;
}
```

결과.

C# Microsoft Visual Studio 디버그 콘솔

```
문자를 계속 입력하고 Enter를 누르면 >>
java
java
python
python
q

문자를 누를 때마다 두 번 출력 >>
jjaavvaag
문자를 누르면 한 번 출력>>
java
```

• 실습예제 5. stringput.c

- 함수 scanf()는 공백으로 구분되는 하나의 문자열을 입력한다.
 - * 함수 scanf("%s" ,str)에서 형식제어문자 %s를 사용한다.
 - * 문자열 입력은 충분한 공간의 문자배열이 있어야 가능하다.
 - * 단순히 문자 포인터로는 문자열 저장이 불가능하다.
- 가장 먼저 입력 받은 문자열이 저장될 충분한 공간인 문자 배열 str을 선언한다.
- 함수printf("%s" ,str)에서 %s를 사용하여 문자열을 출력한다.
- 이름과 성을 분리하여 입력한다면 성만 name[]에 저장
- 함수 printf()에서 %10s는 폭이 10, 우측정렬로 문자열을 출력한다.

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

int main(void)
{
    //char *name, *dept; 실행 오류 발생
    char name[20], dept[30];

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

    return 0;
}
```

결과.

```
Microsoft Visual Studio 디버그 콘솔
학과 입력 >>컴퓨터정보공학과
이름입력 >>김현준
출력: 컴퓨터정보공학과      김현준
```

• 실습예제 6. gets.c

- 함수 gets()와 gets_s()를 사용하여 여러 줄을 입력 받아 출력한다.
- while문을 사용하면 연속된 여러 행을 입력 받아 바로 행 별로 출력
 - * 다음 반복을 종료하려면 새로운 행 처음에 (ctrl + z)를 입력한다.
- 함수 printf()와 scanf()
 - * 다양한 입출력에 적합하다.
- 함수 puts()와 gets()
 - * 처리속도가 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;
}
```

결과.

```
Microsoft Visual Studio 디버그 콘솔
입력을 종료하려면 새로운 행에서 (ctrl + z)를 누르십시오.
문자열 처리를 배우고 있습니다.
문자열 처리를 배우고 있습니다.
Z

gets()의 사용도 마찬가지입니다.
gets()의 사용도 마찬가지입니다.
Z
```


2주차 수업 문자와 문자열

문자열 관련 함수

• 함수 strcat()

- 하나의 문자열 뒤에 다른 하나의 문자열을 연이어 추가해 연결
- 앞 문자열에 뒤 문자열의 null 문자까지 연결한다.
 - * 앞의 문자열 주소를 반환한다.
- 앞 인자인 dest의 저장공간이 연결된 문자열의 길이보다 부족하면 문제가 발생한다.
- 전달인자의 마지막에 연결되는 문자의 수를 지정한다.
- 마지막 수는 널 문자를 제외한 수 이다.

• 함수 strtok()

- 문자열에서 구분자인 문자를 여러 개 지정하여 토큰을 추출하는 함수.
- 첫 번째 인자인 str은 토큰을 추출할 대상인 문자열
- 두 번째 인자인 delim은 구분자로 문자의 모임인 문자열
- 첫 번째 인자인 str은 문자배열에 저장된 문자열을 사용한다.
 - * str은 문자열 상수를 사용 불가능 하다.

앞 문자열 str에서 뒤 문자열 delim을 구성하는 구분자를 기준으로 순서대로 토큰을 추출하여 반환하는 함수이며, 뒤 문자열 delim은 수정될 수 없다.

```
char * strtok(char * str, const char * delim);
```

마지막 인자인 context는 함수 호출에 사용되는 위치 정보를 위한 인자이며, Visual C++에서는 앞으로 함수 strtok_s()의 사용을 권장.

```
char * strtok(char * str, const char * delim, char ** contrxt);
```

• 함수 strtok()

- NULL 문자를 제외한 문자열 길이를 반환하는 함수이다.

• 함수 `strlwr()`

-인자를 모두 소문자로 변환하여 반환하는 함수이다.

• 함수 `strupr()`

-인자를 모두 대소문자로 변환하여 반환하는 함수이다.

여러 문자열 처리

• 문자 포인터 배열

-여러 개의 문자열을 처리하는 하나의 방법이다.

-하나의 문자 포인터가 하나의 문자열을 참조 가능하다.

-문자 포인터 배열은 여러 개의 문자열을 참조 가능하다.

장 단점

-문자 포인터 배열 이용 방법은 각각의 문자열 저장을 위한 최적의 공간을 사용한다.

-문자 포인터를 사용해서는 문자열 상수의 수정은 불가능 하다.

* 문장 `pa[0][2] = 'V'` ;와 같이 문자열의 수정은 실행 오류 예)

```
char *pa[] = {"JAVA", "C#", "C++"}, //배열의 크기는 문자열 개수인 3을 지정하거나 빈 공백으로 한다.  
// 각각의 3개의 문자열 출력  
printf("%s ", pa[0]); printf("%s ", pa[1]); printf("%s\n", pa[2]);
```

• 이차원 문자 배열

-문자의 이차원 배열을 이용하는 방법이다.

-이차원 배열의 열 크기는 문자열 중에서 가장 긴 문자열의 긴 문자열의 길이보다 1 크게 지정한다.

* 가장 긴 문자열 “java” 보다 1이 큰 5를 2차원 배열의 열 크기로 지정 한다.

-이차원 배열의 행의 크기는 문자열 수 이다.

* 3으로 지정 하고나 공백으로 비워둔다.

장 단점

-문자의 이차원 배열에서 모든 열 수가 동일하게 메모리에 할당한다.

* 열의 길이가 서로 다른 경우에는 ‘\0’ 문자가 들어가 낭비된다.

-문자열을 수정 가능하다.

* `ca[0][2] = 'v'` ; 와 같이 원하는 문자 수정이 가능하다.

예)

//첫 번째(행)크기는 문자열 갯수를 지정하거나 빈 공백으로 두며, 두 번째 크기는 문자열 중에서 가장 긴 문자열의 길이보다 1크게 지정한다.

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

//각각의 3개 문자열 출력

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

• 명령행 인자

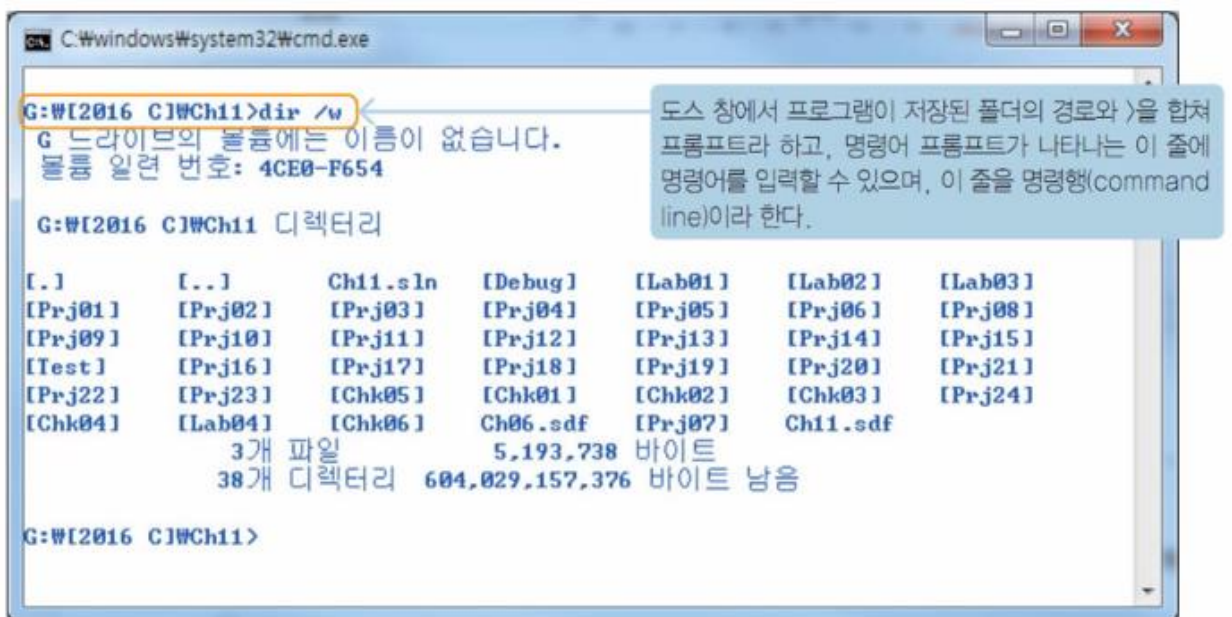
Main(int argc, char*argv[])

-프로그램 dir를 개발한다면 옵션에 해당하는 “/w” 를 어떻게 인식하는가?

- * 명령행 인자를 사용하는 방법
- * 명령행에서 입력하는 문자열을 프로그램으로 전달하는 방법

-프로그램에서 명령행 인자를 받으려면

- * 두 개의 인자 argc와argv를 (int argc, char*argv[])로 기술
- * 매개변수 argc는 명령행에서 입력한 문자열의 수
- * argv[]는 명령행에서 입력한 문자열을 전달 받는 문자 포인터 배열
- * 실행 프로그램 이름도 하나의 명령행 인자에 포함



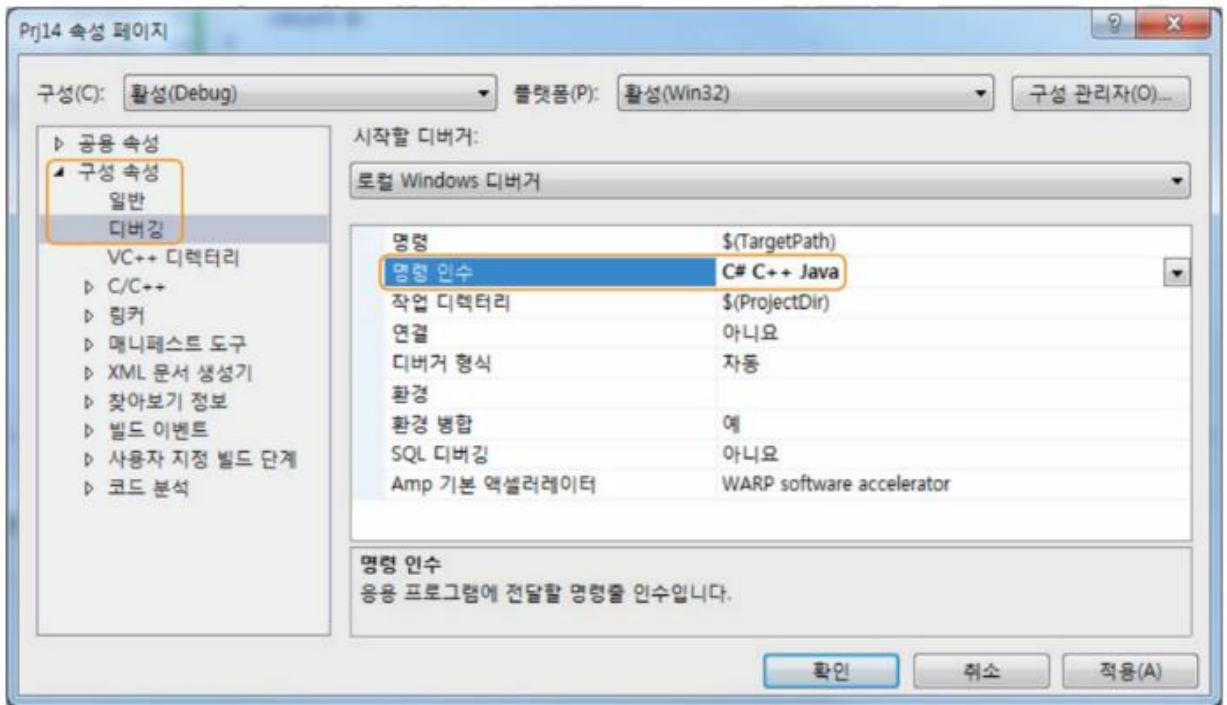
• 명령행에서 실행파일의 이름이 commandarg

-옵션으로 C# C++ Java: 프로그램을 실행한 결과

-명령행 인자로 프로그램을 실행하면 다음과 같은 구조의 문자열 전달

• 명령행 인자 설정

- Visual C++에서 명령행 인자를 설정한다.
- 메뉴 [프로젝트/{프로젝트이름} 속성...]를 누르거나,
단축 키 Alt+F7을 눌러 다음 대화상자에서 설정
 - * 대화상자 [{프로젝트이름} 속성 페이지]의 항목 [디버깅]을 누르고
중간의 [명령 인수]의 입력 상자에 인자를 기술
 - 이 입력상자에서 실행파일 이름 뒤의 옵션만을 기술



2주차 수업 변수 유효범위

전역변수와 지역변수

• 변수의 유효 범위 scope

- 변수의 참조가 유효한 범위
- 변수의 유효 범위 구분
 - * 지역 유효 범위와 전역 유효 범위로 나뉨
- 지역 유효 범위
 - * 함수 또는 블록 내부에서 선언되어 그 지역에서 변수의 참조가 가능한 범위
- 전역 유효 범위
 - * 파일에서만 변수의 참조가 가능한 범위
 - * 프로젝트를 구성하는 모든 파일에서 변수의 참조가 가능한 범위

국내 전용 카드가 지역변수 , 국내외 사용 카드는 전역변수

• 지역변수

- 함수 또는 블록에서 선언된 변수
 - * 내부변수 또는 자동변수라고도 부른다.
- 선언 문장 이후에 함수나 블록의 내부에서만 사용이 가능하다.
 - * 다른 함수나 블록에서는 사용 불가능
- 함수의 매개변수도 함수 전체에서 사용 가능한 지역변수
- 선언 후 초기화하지 않으면 쓰레기값이 저장되므로 주의
- 변수가 선언된 함수 또는 블록에서 선언 문장이 실행되는 시점에서 메모리에 할당한다

• 스택 stack

- 지역변수가 할당되는 메모리 영역
- 선언된 부분에서 자동으로 생성되고 함수나 블록이 종료 되는 순간 메모리에서 자동으로 제거된다.
- 지역변수는 자동변수라 부른다.
- 지역변수 선언에서 자료형 앞에 키워드 auto가 사용될 수 있다.
- 키워드 auto는 생략 가능 하여 일반적으로 auto가 없는 경우가 대부분 이다.

• 전역변수

- 함수 외부에서 선언되는 변수
- 외부 변수라고도 부른다.
- 일반적으로 프로젝트의 모든 함수나 블록에서 참조 가능
- 선언되면 자동으로 초기값이 자료형에 맞는 0으로 지정
 - * 즉 정수형은 0, 문자형은null 문자인 ‘\0’ ; 실수형은 0.0
- 포인터 형은 NULL 값이 저장
- 함수나 블록에서 전역변수와 같은 이름으로 지역변수를 선언 가능
- 함수내부나 블록에서 그 이름을 참조하면 지역변수로 인식
 - * 그러므로 지역변수와 동일한 이름의 전역변수는 참조 불가능]
 - * 가능한 이러한 변수는 사용하지 않도록 한다.
- 전역변수는 동일 프로젝트의 다른 파일에서도 참조가 가능
- 다른 파일에서 선언된 전역변수를 참조하려면 키워드 extern을 사용하여 이미 다른 파일에서 선언된 전역변수임을 선언
- extern을 사용한 참조선언 구문 : 변수선언 문장 맨 앞에 extern을 넣는 구조 – extern 참조선언 구문에서 자료형은 생략 가능

키워드 extern을 사용한 변수 선언은 새로운 변수를 선언하는 것이 아니며 단지 이미 존재하는 전역변수의 유효 범위를 확장

• 전역변수 장단점

- 전역변수의 선언 위치가 변수를 참조하려는 위치보다 뒤에 있는 경우 전역변수를 사용하기 위해서는 extern을 사용한 참조선언이 필요.
 - * 동일한 파일에서도 extern을 사용해야 하는 경우가 발생가능
 - * 소스 파일 중간이나 하단에 전역변수를 배치하는 방법은 바람직하지 않는다.
- 전역변수는 어디에서든지 수정할 수 있으므로사용이 편한 장점이 있다.
- 전역변수에 예상하지 못한 값이 저장된다.
 - * 프로그램 어느 부분에서 수정되었는지 알기 어려운 단점이 있다.
 - * 이러한 문제로 전역변수는 가능한 제한적으로 사용하는것이 바람직

• 정적 변수의 초기화

- 예1 소스에서 정적 변수의 초기값에 변수를 대입
 - * 초기화 문법 오류가 발생
- 예2 와 같이 상수를 대입

예1)

```
#include <stdio.h>

int a = 1;
static s = a; //오류

int main(void)
{
    int data = 10;
    static value = data; //오류

    return 0;
}
```

예2)

```
#include <stdio.h>

int a = 1;
static s = 1;

int main(void)
{
    int data = 10;
    static value = 10;
    |
    return 0;
}
```

정적 변수와 레지스터 변수

• 기억부류

- 변수 선언의 위치에 따라 변수 : 전역과 지역
- 변수 4가지 기억부류 auto, reister, static, extern
 - * 할당되는 메모리영역이 결정되고 메모리의 할당과 제거 시기가 결정

- 기억 부류는 키워드 auto, register, static, extern에 의해 구분
 - * 자동변수인 auto는 일반 지역변수로 생략 가능하다.
- 전역 변수 또는 지역변수
- auto 와 register
 - * 지역변수에만 이용이 가능하다.
- static
 - * 지역 전역 모든 변수에 이용 가능하다.
- extern
 - * 전역변수에만 사용이 가능하다.
 - * 컴파일러에게 변수가 이미 어딘가 (주로 다른 파일)에 존재하고 이제 사용하겠다는 것을 알리는 구문에 사용되는 키워드 이다.
 - * extren이 선언되는 위치에 따라 이 변수의 사용의 범위는 전역 또는 지역으로 한정된다.
- 기억부류 auto, register, static
 - * 새로운 변수의 선언에 사용되는 키워드 이다.
- 기억부류 사용 구문
 - * 변수 선언 문장에서 자료형 앞에 하나의 키워드를 넣는 방식
 - * 키워드 extern을 제외하고 나머지 3개의 기억부류의 변수선언에서 초기값을 저장 가능하다.
 - * 이미 지역 변수에서 다룬 것처럼 키워드 auto는 지역변수 선언에 사용되며 생략 가능하다.
 - * 함수에 선언된 모든 변수가 auto가 생략된 자동변수

•키워드 register

- 변수를 연산에 참여 시키려면 다시 CPU 내부에 레지스터에 불러들여 연산을 수행한다.
- 레지스터 변수이다.
- 변수의 저장공간이 이미 메모라가 아니라 CPU 내부의 레지스터에 할당되는 변수이다.
- 키워드 register를 자료형 앞에 넣어 선언한다.
- 지역변수에만 이용이 가능하다.
- 지역변수로서 함수나 블록이 시작되면서 CPU의 내부 레지스터에 값이 저장된다.
 - * 함수나 블록을 빠져나오면서 소멸되는 특성이 있다.
- CPU내부에 있는 기억장소이므로 일반 메모리보다 빠르게 참조 가능.
- 일반 메모리에 할당되는 변수가 아니므로 주소연산자 &를 사용 불가능
 - * 주소연사자 &를 사용하면 문법오류가 발생.

• 키워드 static

- 정적변수 (static variable)를 선언
 - * 변수 선언에서 자료형 앞에 키워드 static 기송
- 정적 지역변수 와 정적 전역변수로 구분 한다.
- 초기 생성된 이후 메모리에서 제거되지 않는다.
 - * 지속적으로 저장값을 유지하거나 수정 가능한 특성
 - * 프로그램이 시작되면 메모리에 할당되고, 프로그램이 종료되면 메모리에서 제거된다.
- 초기값
 - * 초기값을 지정하지 않으면 자동으로 자료형에 따라 0 이나 ‘\0’ 또는 NULL 값이 저장.
 - * 초기화는 단 한번만 수행한다.
 - * 한번 초기화된 정적변수는 프로그램 실행 중간에 더 이상 초기화되지 않는 특성이 있다.
 - * 주의 : 초기화는 상수로만 가능하다.

2주차 수업 과제

교재 11장 프로그래밍 연습 01에서 10까지 중 5문제 프로그래밍 구현

• 프로그래밍 연습 1.

한 행을 표준 입력으로 입력 받은 문자열의 길이를 구하는 함수 mystrlen()을 구현하여 라이브러리 strlen()과 결과를 비교하는 프로그램을 작성 하시오.

- * int mystrlen(const char*p)
- * 한 행을 표준 입력으로 입력 받는 것은 라이브러리 gets()사용


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

int mystrlen(const char* p);

int main(void)
{
    char str[100];
    puts("문자열을 입력하세요");
    gets(str);
    printf("문자열 길이 mystrlen() 사용: %d\n", mystrlen(str));
    printf("문자열 길이 strlen() 사용: %d\n", strlen(str));
    return 0;
}
```



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

 Microsoft Visual Studio 디버그 콘솔

```
문자열을 입력하세요
phantom or tanatos
문자열 길이 mystrlen() 사용: 18
문자열 길이 strlen() 사용: 18
```

• 프로그래밍 연습 2.

앞의 문자열에 뒤 문자열을 연결하는 함수 mystrcat()를 구현하여 다음을 예로 함수 mystrcat()의 결과를 출력하는 프로그램을 작성하시오.

- * void mystrcat(char s1[], const char s2[]):라이브러리 strcat()와 같이 s1 뒤에 s2를 붙여 연결하는 함수

```
char s1[50] = "C ";
mystrcat(s1, "programming language" );
```

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

void mystrcat(char s1[], const char s2[]);

int main(void)
{
    char s1[50] = "C ";
    mystrcat(s1, "programming language");
    printf("%s\n", s1);
    return 0;
}

void mystrcat(char s1[], const char s2[])
{
    int i = strlen(s1);
    int j = 0;
    while (s2[j] != '\0' && i < 50) {
        s1[i] = s2[j];
        ++i;
        ++j;
    }
    s1[i] = '\0';
}
```

C programming language

C:\Users\win\source\repos\20193418\Debug\02.exe(프로세스 7548개)이(가) 디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] 하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...

• 프로그래밍 연습 3.

앞의 문자열에서 뒤 문자를 삭제하는 함수 `delchar()`를 구현하여 다음을 예로 함수 `delchar()`의 결과를 출력하는 프로그램을 작성하시오.

- * `void delchar(char str[], const char ch)`:str에서 문자 `ch`를 삭제한 문자열을 반영하는 함수
- * 다음 변수를 사용하며, 라이브러리 `strcpy()`를 사용하여 문자배열 `str`에 문자열 “java”를 저장
- * 문자열 `str`에서 문자 `ch`를 삭제하도록 `delchar()`를 호출

`char str [20];`

`char ch = 'a' ;`

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

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

int main(void) {
    char str[20];
    char ch = 'a';

    strcpy(str, "java");

    delchar(str, ch);
}

void delchar(char str[], const char ch) {
    int count = 0;

    while (str[count] != NULL) {
        if(str[count] == ch)
            for (int i = count; str[i] != NULL; i++) {
                str[i] = str[i + 1];
            }
        count++;
    }
    printf("ch에 해당하는 값을 제거 후 str의 값 = %s\n", str);
}
```

Microsoft Visual Studio 디버그 콘솔

ch에 해당하는 값을 제거 후 str의 값 = jv

C:\Users\win\source\repos\20193418\Debug\04.exe(프로세스 17204개)
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] 탭에서 콘솔을 자동으로 닫으려는 옵션을 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...

• 프로그래밍 연습 4.

한 단어를 표준입력으로 입력 받아 각각의 단어를 구성하는 문자를 역순으로 출력하는 프로그램을 작성하시오.

한 단어를 입력하세요. - > programming

입력한 단어를 반대로 출력합니다. gnimmargorp

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

int main(void) {
    char arr[25];

    printf("단어 입력:");
    gets_s(arr, sizeof(arr));

    int size = strlen(arr);

    puts("입력받은 단어를 역순으로 출력");
    for (int i = size; i >= 0; i--) {
        printf("%c", arr[i]);
    }
    puts("");
}
```

Microsoft Visual Studio 디버그 콘솔

단어 입력:programming
입력받은 단어를 역순으로 출력
gnimmargorp

C:\Users\win\source\repos\20193418\Debug\06.exe(프로세스 2205)
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션]
1를 사용하도록 설정합니다.

• 프로그래밍 연습 5.

한 줄의 문자열을 표준입력으로 입력 받아 단어의 문자를 역순으로 출력하는 프로그램을 작성하시오.

한 줄의 문장을 입력 하세요. ->


I' ve compiled with c++ powerpoint presentation

입력한 각각의 단어를 반대로 출력합니다. ->

ev' I delipmoc htiw ++c tnioprewop noitatneserp

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>
void reverse(char* word);
int main(void)
{
    char line[100];
    char* token;
    printf("한 줄의 문장을 입력하세요 ->\n");
    scanf("%[^\n]s", line);
    printf("입력한 각각의 단어를 반대로 출력합니다 ->\n ");
    token = strtok(line, " ");
    while (token != NULL) {
        reverse(token);
        printf("%s ", token);
        token = strtok(NULL, " ");
    }
    printf("\n");
    return 0;
}

void reverse(char* word) {
    int i, size = strlen(word);
    char temp;
    for (i = 0; i < size / 2; i++) {
        temp = word[i];
        word[i] = word[(size - 1) - i];
        word[(size - 1) - i] = temp;
    }
}
```

 Microsoft Visual Studio 디버그 콘솔

한 줄의 문장을 입력하세요 ->

i've compiled with c++ powerpoint presentation

입력한 각각의 단어를 반대로 출력합니다 ->

ev'i delipmoc htiw ++c tnioprewop noitatneserp

C:\Users\win\source\repos\20193418\Debug\08.exe(프로세스 18660개)
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [

3주차 수업 변수 유효범위

메모리 영역과 변수 이용

• 메모리 영역

- 데이터, 스택, 힙 영역
 - * 메모리 영역은 변수의 유효범위와 생존기간에 결정적 역할
- 변수는 기억부류에 따라 할당되는 메모리 공간이 달라진다.
 - * 기억부류는 변수의 유효와 생존기간을 결정한다.
 - * 기억부류는 변수의 저장공간의 위치데이터 영역, 힙 영역, 스택 영역인지도 결정하며, 초기값도 결정한다.

• 데이터 영역

- 전역변수와 정적변수가 할당되는 저장공간 이다.
- 메모리 주소가 낮은 값에서 높은 값으로 저장 장수가 할당된다.
- 프로그램이 시작되는 시점에 정해진 크기대로 고정된 메모리 영역이 확보된다.

• 힙 영역

- 동적 할당되는 변수가 할당되는 저장공간 이다.
- 데이터 영역과 스택 영역 사이에 위치해 있다.

• 스택 영역

- 함수 호출에 의한 형식 매개변수 그리고 함수 내부의 지역변수가 할당되는 저장공간 이다.
- 힙 영역과 스택영역은 프로그램이 실행되면서 영역 크기가 계속적으로 변한다.
- 메모리 주소가 높은 값에서 낮은 값으로 저장 장소가 할당된다.
 - * 함수 호출과 종료에 따라 메모리가 할당되었다가 다시 제거되는 작업이 반복된다.

• 변수 이용 기준

- 전역변수의 사용을 자제하고 지역변수를 주로 이용한다.
- 레지스터 변수
 - * 실행 속도를 개선하고자 하는 경우
- 정적 지역변수
 - * 함수나 블록 내부에서 함수나 블록이 종료되더라도 계속적으로 값을 저장한다.
- 정적 전역변수
 - * 해당 파일 내부에서만 변수를 고유하고자 하는 경우
- 전역변수
 - * 프로그램의 모든 영역에서 값을 공유하고자 하는 경우
 - * 가능하면 전역 변수의 사용을 줄이는 것이 프로그램의 이해를 높일 수 있으며 발생할 수 있는 프로그램 문제를 줄일 수 있음

• 전역변수와 지역변수를 정리

- 변수 할당 메모리 영역에 따라 변수의 할당과 제거의 시기가 결정
- 데이터 영역의 전역변수와 정적 변수
 - * 프로그램 시작 시 메모리가 할당되고, 프로그램 종료 시 메모리에서 제거
- 스택 영역과 레지스터에 할당되는 자동 지역변수와 레지스터 변수
 - * 함수 또는 블록 시작 시 메모리가 할당되고, 함수 또는 블록 종료 시 메모리에서 제거

• 변수의 유효범위와 초기값

- 변수의 유효 범위는 O, X로 구분
 - * 그 지역에서 참조 가능하면 O 아니면 X로 구분한다.
- 변수의 종류에 따라 초기값
 - * 초기 값 저장 문장의 실행 시점
 - * 초기값이 명시적으로 지정되지 않을 경우 자동으로 지정되면 기본 초기값

과제

교재 12장 LAB 12-1, 2, 3의 구현, 코딩과 결과를 제출.

• LAB 1. 피보나츠 수의 출력

- 피보나츠의 수는 1, 1로 시작하여 이전 두 수를 더한 수
- 5개의 피보나츠 수는 1, 1, 2, 3, 5
- 표준입력으로 받은 3 이상의 정수를 전역변수 count에 저장한 후 재귀함수인 fibonacci()에서 count-1 개의 피보나츠의 수를 출력
- 함수 fibonacci()의 매개변수는 (int prev_number, int number)으로 이전 두 정수가 인자, 자기자신을 호출하는 재귀함수
- 함수 fibonacci()에서 자기 자신이 호출된 수를 저장하는 정적 지역 변수 i를 사용하며 초기값은 1로 지정

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

//전역변수
int count;
//함수원형
void fibonacci(int prev_number, int number);

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

    printf("피보나츠를 몇 개 구할까요?(3 이상) >> ");
    //전역변수를 표준입력으로 저장
    scanf_s("%d", &count);
    if (count <= 2)
        return 0;
    printf("1 ");
    fibonacci(prev_number, number);
    printf("\n");
}

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

    //전역변수 count와 함수의 정적 지역변수를 비교
    while (i++ < count)
    {
        //지역변수
        int next_num = prev_number + number;
        prev_number = number;
        number = next_num;
        printf("%d ", next_num);
        fibonacci(prev_number, number);
    }
}
```

```
피보나츠를 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
```

• LAB 2. 지역변수와 정적변수의 사용

-함수 main()에서 함수 process()를 세 번 호출

- * 함수 process()에서 지역변수와 정적 지역변수가 선언
- * 간단한 연산과 함께 출력

```
#include <stdio.h>

void process();

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

    return 0;
}

void process()
{
    //정적 변수
    static int sx;
    //지역 변수
    int x = 1;

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

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

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


• LAB 3. 은행계좌의 입출금 구현

- 전역변수 total, 두 함수 save()와 withdraw() 구현
- 정적 지역 변수 amount를 사용
- 몇 개의 입출금에 대해 출력
 - * 전역변수 total에는 초기 금액과 계좌 잔고가 저장
- 함수 save()와 withdraw()는 각각 매개변수 금액의 입출금을 구현
 - * 정적 지역변수 amount를 사용하여 총입 금액과 총출금액을 관리하여 출력한다.

```
#include<stdio.h>

//전역변수
int total = 10000;
//입금 함수원형
void save(int);
//출금 함수원형
void withdraw(int);

int main(void) {
    printf(" 입금액   출금액   총입금액   총출금액   잔고\n");
    printf("===== \n");
    printf("%46d\n", total);
    save(50000);
    withdraw(30000);
    save(60000);
    withdraw(20000);
    printf("===== \n");

    return 0;
}

//입금액을 매개변수로 사용
void save(int money) {
    //총입금액이 저장되는 정적 지역변수
    static int amount;
    total += money;
    amount += money;
    printf("%7d %17d %20d\n", money, amount, total);
}

//출금액을 매개변수로 사용
void withdraw(int money) {
    //총출금액이 저장되는 정적 지역변수
    static int amount;
    total -= money;
    amount += money;
    printf("%15d %20d %9d\n", money, amount, total);
}
```

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

4주차 수업 구조체와 공용체

구조체와 공용체 개념

• 구조체 개념

- 정수, 문자, 실수나 포인터 그리고 이들의 배열 등을 묶어 하나의 자료형으로 이용하는 것
 - * 선물세트 - 인기가 있거나 관련 있는 상품들을 묶어 하나의 구성제품으로 판매하는 것 과 같은 개념
- 서로 관련 있는 정보들을 하나로 묶어 처리하는 경우가 흔히 발생한다.
- 차에 대한 정보 계좌에 대한 정보, 책에 대한 정보, 학생 교수, 강좌에 대한 정보
- C 언어는 이러한 요구사항을 구조체(struct)로 지원
 - * 연관성이 있는 서로 다른 개별적인 자료형의 변수들을 하나의 단위 단위로 묶은 새로운 자료형
 - * 멤버로 구성되는 통합 자료형으로 대표적인 유도 자료형

• 구조체 정의

- 와플이나 봉어빵을 만들려면 기계가 필요하듯이 구조체를 자료형으로 사용하려면 먼저 구조체를 정의
- 구조체 틀을 만드는 구조체 정의 방법
 - *키워드 struct 다음에 구조체 태그 이름을 기술 중괄호를 이용하여 원하는 멤버를 여러 개의 변수로 선언하는 구조
- 구조체 멤버 또는 필드는 구조체를 구성하는 하나 하나의 항목

•구조체 정의 구문

-구조체 정의는 변수의 선언과는 다름

* 변수선언에서 이용될 새로운 구조체 자료형을 정의하는 구문

-모두 하나의 문장이므로 반드시 세미콜론으로 종료

-각 구조체 멤버의 초기값 대입 불가능

-모든 멤버 선언에 반드시 세미콜론삽입, 마지막 멤버도;

-int credit; int hour;, int credit, hour; 로도 가능

-구조체 멤버의 이름은 모두 유일

-멤버로는 다양한 자료형, 다른 구조체 변수 및 구조체 포인터도 허용

•구조체 태그이름: account

-struct account : 계좌정보를 표현하는 구조체

* 계좌주 이름, 계좌번호, 잔고 정보를 하나의 단위로 처리하는 자료형을 정의

예)

```
struct account
{
    char name[10]; //계좌주 이름 한글일경우 5글자만 가능
    int acnum;      //계좌번호
    double balance; //잔고
};
```

•구조체 변수 선언

-구조체가 정의되었다면 구조체형 변수 선언이 가능하다.

* 구조체 struct account가 새로운 자료유형으로 사용 가능

-새로운 자료형 struct account 형 변수 mine을 선언 구문

* struct account mine;

-구조체 정의와 변수 선언을 함께하는 방법은 문장 이후 struct account도 새로운 자료형으로 사용 가능

예)

```
struct account
{
    char name[12]; //계좌주이름
    int acnum;      //계좌번호
    double balance; //잔고
}myaccount;

struct account youraccount;
```

•구조체 변수의 초기화

-변수 선언 시 중괄호를 이용한 초기화 지정이 가능하다.

- * 초기화 값은 중괄호 내부에서 각 멤버 정의 순서대로 초기화 값을 구분하여 기술한다.
- * 기술되지 않은 멤버값은 자료형에 따라 기본값인 0,0.0,'\0'등으로 저장 된다.

예)

```
struct account
{
    char name[12]; //계좌주이름
    int acnum;      //계좌번호
    double balance; //잔고
};

struct account mine = {"홍길동", 100, 30000};
```

-구조체 태그이름이 없는 구조체변수 선언 구문

- * 이 구조체와 동일한 자료형의 변수를 더 이상 선언 불가능
- * 단 한번 이 구조체 형으로 변수를 선언하는 경우에만 이용
- * 단 이러한 태그이름이 없는 구조체 정의 바로 변수가 나오지 않는다면 아무 의미 없는 문장이다.

예)

```
struct account
{
    char name[12]; //계좌주이름
    int acnum;      //계좌번호
    double balance; //잔고
}youraccount;
```

•구조체의 멤버 접근 연산자 . 와 변수 크기

-선언된 구조체형 변수에서 멤버 접근 방법

-접근연산자 .를 사용하여 멤버를 참조

-문장 yours.actnum=1002;

- * 변수 yours의 멤버 actnum에 1002를 저장하는 기능을 수행
- * 접근연산자는 .는 참조 연산자라고도 부른다.

-구조체 struct account의 변수 mine은 다음 구조로 메모리에 할당 된다.

- * 변수 mine의 크기는 sizeof(mine)로 가능
- * 실제 구조체의 크기는 멤버의 크기의 합보다 크거나 같을 수 있음.

•구조체 멤버로 사용되는 구조체

-구조체 멤버로 가능

- * 이미 정의된 다른 구조체 형 변수
- * 자기 자신을 포함한 구조체 포인터 변수

-구조체 struct date

- * 년, 월, 일 정보를 저장할 수 있는 구조체

-구조체 struct account

- * 계좌 개설일자를 저장할 멤버로 open을 추가
- * open의 자료형으로 위에서 정의한 struct date를 사용한다.
- * struct account 변수 me의 메모리 구조

•구조체 정의 위치

-구조체 정의는 그 정의 위치에 따라 구조체의 유효 범위가 결정

-구조체의 정의도 변수 선언처럼 유효범위는 전역 또는 지역

-전역 : main() 함수 외부 상단에서 정의된 구조체

-지역: main() 함수 또는 다른 함수 내부에서 정의된 구조체

•구조체 변수의 대입

-동일한 구조체형의 변수는 대입문이 가능하다.

-변수 대입으로 한번에 모든 멤버의 대입이 가능하다.

•구조체의 동등 비교

-struct student형의 변수 hong과 one 에서 (one == bae)

- * 동등 비교는 사용 불가능 하다.

-만일 구조체를 비교하려면 구조체 멤버, 하나 하나를 비교한다.

•char 포인터

-문자열의 첫 문자 주소를 저장하므로 문자열 상수의 주소로 사용

•char 배열

-문자열을 구성하는 모든 문자를 하나 하나 저장하고 마지막에

‘\0’ 문자를 저장하여 사용한다.

•공용체 개념

- 하나의 차고에 일반 세단과 SUV를 각각 주차 한다고 생각 하는 개념
- 이러한 겸용 주차장과 비슷한 개념이다.
- 동일한 저장 장소에 여러 자료형을 저장하는 방법
- 공용체를 구성하는 멤버에 한번에 한 종류만 저장하고 참조 가능하다.

•union을 사용한 공용체 정의 및 변수 선언

- 공용체(union)
 - * 서로 다른 자료형의 값을 동일한 저장공간에 저장하는 자료형
- 공용체 선언 방법
 - * union을 struct로 사용하는 것을 제외하면 구조체선언 방법과 동일하다.

•공용체 크기와 초기화

공용체 변수의 크기

- 멤버 중 가장 큰 자료형의 크기로 정해진다.
- union data의 변수 data1은 멤버 중 가장 큰 크기인 double 형의 8바이트의 저장공간을 세 멤버가 함께 이용한다.
- 동시에 여러 멤버의 값을 동시에 저장하여 이용할 수 없으며 마지막에 저장된 단 하나의 멤버 자료값만을 저장한다.
- 공용체도 구조체와 같이 typedef를 이용하여 새로운 자료형으로 정의 가능하다.

공용체의 초기화

- 공용체 정의 시 처음 선언한 멤버의 초기값으로만 저장이 가능하다.
만일 다른 멤버로 초기값을 지정하면 컴파일 시 경고가 발생한다.
 - * 초기값으로 동일한 유형의 다른 변수의 대입도 가능하다.

•공용체 멤버 접근

- 구조체와 같이 접근연산자 .를 사용한다.
- 문장 data2.ch= 'A' ;
 - * 이 문장 이후에 멤버 cnt나 real의 출력은 가능하나 의미는 없다.
- 유형이 char인 ch를 접근하면 8바이트 중에서 첫 1바이트만 참조
 - * int인 cnt를 접근하면 전체 공간의 첫 4바이트만 참조
 - * double인 real을 접근하면 8바이트공간을 모두 참조
- 항상 마지막에 저장한 멤버로 접근해야 원하는 값을 얻을 수 있다.

자료형 재정의

• 자료형 재정의 typedef

-typedef 구문

- * typedef는 이미 사용되는 자료 유형을 다른 새로운 자료형 이름으로 재정의할 수 있도록 하는 키워드 이다.

-typedef int profit;

- * profit을 int 와 같은 자료형으로 새롭게 정의하는 문장

• 자료형을 재정의하는 이유

-프로그램의 시스템 간 호환성과 편의성을 위해 필요

-터보 C++ 컴파일러에서 자료유형 int는 저장공간 크기가 2바이트이다.

- * Visual C++에서는 4바이트

- * Visual C++에서 작성한 프로그램은 터보 C++에서는 문제가 발생

-Visual C++에서는 다음과 같이 int를 myint로 재정의

-모든 int 형을 myint형으로 선언하여 이용한다.

• struct를 생략한 새로운 자료형

-typedef 사용하여 구조체 struct date를 date로 재정의

-물론 date가 아닌 datatype등 다른 이름으로도 재정의가 가능

-typedef 구문에서 새로운 자료형으로 software 형이 정의

-이 구문 이후에는 software를 구조체 자료형으로 변수 선언에 사용

-구조체 태그이름은 생략 가능

-구조체 software 형은 멤버로 구조체 date형 변수 release

과제

교재 13장 실습예제 13-1, 2, 3, 4의 구현, 코딩과 결과를 제출하시오.

• 실습예제 1. structbasic.c

구조체 정의화 선언

-은행계좌를 위한 구조체 사용

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

//은행 계좌를 위한 구조체 정의
struct acct
{
    char name[12]; //계좌주 이름
    int actnum; //계좌번호
    double balance; //잔고
};

int main (void)
{
    //구조체 변수 선언 및 초기화
    struct acct mine = { "홍길동", 1001, 300000 };
    struct acct 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: 24홍길동 1001 300000.00
이동원 1002 500000.00
```


• 실습예제 2. nestedstruct.c

구조체 멤버의 구조체

- account 구조체를 사용한 프로그램
- 멤버가 구조체 date인 초기화 {2012, 3, 9}
- 구조체 account 변수인 me로 년, 월, 일을 참조
 - * 접근연산자를 2번 사용

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

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

//은행계좌를 위한 구조체
struct account
{
    struct date open; //계좌 개설일자
    char name[12]; //계좌주 이름
    int actnum; //계좌번호
    double balance; //잔고
};

int main(void)
{
    struct account me = { { 2018, 3, 9 }, "홍길동", 1001, 30000 };

    printf("구조체크기: %d\n", sizeof(me));
    printf("[%d, %d %d]\n", me.open.year, me.open.month, me.open.day);
    printf("%s %d %.2f\n", me.name, me.actnum, me.balance);
}
```

 Microsoft Visual Studio 디버그 콘솔

```
구조체크기: 40
[2018, 3 9]
홍길동 1001 30000.00
```

• 실습예제 3. structstudent.c

구조체 대입과 비교

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

int main(void)
{
    //학생을 위한 구조체
    struct student
    {
        int snum;        //학번
        char* dept;      //학과 이름
        char name[12];   //학생 이름
    };

    struct student hong = { 201800001, "컴퓨터정보공학과", "홍길동" };
    struct student na = { 201800002 };
    struct student bae = { 201800003 };

    //학생이름 입력
    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 이동합니다.\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;
}
```

Microsoft Visual Studio 디버그 콘솔

```
나한국
[201800001, 컴퓨터정보공학과, 홍길동]
[201800002, 컴퓨터정보공학과, 나한국]
[201800003, 기계공학과, 배상문]
학번이 201800003(으)로 동일합니다.
내용이 같은 구조체입니다.
```

• 실습예제 4. union.c

공용체 정의와 선언

-문자와 정수와 실수를 각각 하나씩 저장할 수 있는 공용체의 정의와 활용

```
#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' };
    union data data3 = data2;

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

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

5주차 수업 구조체와 공용체

구조체와 공용체의 포인터와 배열

• 포인터 변수 선언

-구조체 포인터는 구조체의 주소값을 저장하는 변수

-대학 강좌를 처리하는 구조체

자료형 lecture를 선언한 구문

-구조체 포인터 변수 p는 lecture *p
로 선언

-변수 os를 선언한 후

문장 lecture *p = &os;

- * lecture 포인터 변수 p에
&os를 저장

- * 이로써 포인터 p로 구조체 변수
os멤버 참조가 가능

```
struct lecture
{
    char name[20];    //강좌명
    int type;         //강좌구분
    int credit;       // 학점
    int hours;        //사수
};
typedef struct lecture lecture;
lecture *p;
```

• 포인터 변수의 구조체 멤버 접근 연산자 ->

-P -> name

-포인터 p가 가리키는 구조체 변수의 멤버 name을 접근하는 연산식

-p->type, p->credit, p->hours: 각각 os.type, os.credit ,
os.hours를 참조

- * ->에서 -와 > 사이에 공백이 들어가는는 절대 안된다.

-연산식 (*p).name으로도 사용이 가능하다.

- * (*p).name은 *p.name과는 다르다.

 - *p.name은 *(p.name)과 같은 연산식

 - p가 포인터이므로 p.name 는 문법오류가 발생한다.

• 접근연산자 ->와.의 연산자 우선순위의

-간접연산자 *를 포함한 다른 어떠한 연산자 우선순위보다 가장 높다.

- * 연산자 ->와.은 우선순위 1위이고 결합성은 좌에서 우이다.

- * 연산자 *은 우선순위 2이고 결합성은 우에서 좌이다.

• 구조체 배열 변수 선언

- 구조체 lecture의 배열크기 3인 c를 선언하고 초기화값을 저장하는 구문이다.
 - * 구조체 배열의 초기값 지정 구문에서는 종괄호가 중첩되게 표시
 - * 외부 종괄호는 배열 초기화의 종괄호이며, 내부 종괄호는 배열원소인 구조체 초기화를 위한 종괄호 이다.

5주차 수업 함수와 포인터 활용

함수의 인자전달 방식

• 함수에서 값의 전달

- C 언어는 함수의 인자 전달 방식
 - * 기본적으로 값에 의한 호출 방식
 - * 함수 호출 시 실인자의 값이 형식인자에 복사되어 저장된다는 의미 이다.
- 함수 `increase(int origin, int increment)`
- `origin += increment;` 를 수행하는 간단한 함수
- 함수 `increase()` 내부실행
- 변수 `amoun`와 매개변수 `origin`은 아무 관련성이 없다.
 - * `origi`은 증가해도 `amount`의 값은 변하지 않는다.
- 함수 외부의 변수를 함수 내부에서 수정할 수 없는 특징이 있다.

• 함수에서 주소의 전달

- 함수 `increase()`
 - * 첫 번째 매개변수를 `int*` 로 수정
 - * 함수 구현도 `*origin += increment;` 로 수정하여 구현
- 함수 호출 시 첫 번째 인자가 `&amount`이므로 변수 `amount`의 주소 값이 매개변수인 `origin`에 복사
- 함수 `increase()` 내부실행
 - * `*origin`은 변수 `amount`자체를 의미한다.
 - * `*origin`을 증가시키려면 `amount`의 값도 증가.

-참조에 의한 호출

- * 포인터를 매개변수로 사용하면 함수로 전달된 실인자의 주소를 이용하여 그 변수를 참조 가능하다.
- * 함수에서 주소의 호출

• 배열이름으로 전달

- 함수의 매개변수로 배열을 전달하는 것이다.
- 배열의 첫 원소를 참조 매개변수로 전달하는 것과 동일하다.
- 배열을 매개변수로 하는 함수 `sum()`을 구현
- 실수형 배열의 모든 원소의 합을 구하여 반환하는 함수
- 함수 `sum()`의 형식매개변수는 실수형 배열과 배열크기
- 첫 번째 형식매개변수에서 배열자체에 배열크기를 기술하는 것은 아무 의미가 없다.
 - * `double ary[5]` 보다 `double ary[]`라고 기술하는 것을 권장
 - * 실제로 함수 내부에서 실인자로 전달된 배열의 배열크기를 알 수 없다.
 - * 배열크기를 두 번째 인자로 사용
- 매개변수를 `double ary[]` 처럼 기술해도 단순히 `double *ary`처럼 포인터 변수로 인식한다.

• 배열크기로 인자로 사용

- 만일 배열크기를 인자로 사용하지 않는다면 정해진 상수를 함수정의 내부에서 사용해야 한다.
 - * 이러한 방법은 배열크기가 변하면 소스를 수정해야 하므로 비효율적이다.
- 배열크기에 관계없이 배열 원소의 합을 구하는 함수를 만들려면 배열 크기도 하나의 인자로 사용된다.

• 다양한 배열원소 참조 방법

- 배열 `point`에서 간접연산자를 사용한 배열원소의 접근 방법은
 - * `(point + i)`
- 배열의 합을 구하려면 `sum += *(point + i);` 문자열 반복
- 문장 `int *address = point;`
 - * 배열 `point`를 가리키는 포인터 변수 `address`를 선언하여 `point`를 저장
- 문장 `sum += *(address++)`으로도 배열의 합 가능
- 배열이름 `point`는 주소 상수이다.

• 형식 매개변수 `int ary[]`와 `int*art`

-함수 헤더에 배열을 인자로 기술하는 방법

- * 함수헤더에 `int ary[]`로 기술하는 것은 `int*ary`로도 대체 가능

• 배열크기 계산방법

-배열이 함수인자인 경우

- * 대부분 배열크기도 함수인자로 하는 경우가 일반적이다.

-배열크기

- * `(sizeof(배열이름) / sizeof(배열원소))`

• 다차원 배열 전달

-이차원 배열을 함수 인자로 이용하는 방법

- * 이차원 배열에서 모든 원소의 합을 구하는 함수를 구현
- * 다차원 배열을 인자로 이용하는 경우 첫 번째 대괄호 내부의 크기를 제외한 다른 모든 크기는 반드시 기술

-이차원 배열의 행의 수를 인자로 이용하면 보다 일반화된 함수를 구현 가능하다.

-함수 `sum()`

- * 이차원 배열값을 모두 더하는 함수
- * 함수 `printarray()`는 인자인 이차원 배열값을 모두 출력하는 함수

• 이차원 배열 행과 열

-함수 `sum()`을 호출하려면 배열이름과 함께 행과 열의 수가 필요

-이차원 배열의 행의 수

- * `(sizeof(x) / sizeof(x[0]))`

-이차원 배열의 열의 수

- * `(sizeof(x[0] / sizeof(x[][0][0]))`

-`sizeof(x)`는 배열 전체의 바이트 수 ,`sizeof(x[0])`는 1행의 바이트수

-`sizeof(x[0][0])`은 첫 번째 원소의 바이트 수

• 가변 인자가 있는 함수머리

-함수 `printf()` 함수 원형

- * 첫 인자는 `char*Format`을 제외하고는 이후에...표시

-함수 `printf()`를 호출하는 경우는

- * 출력할 인자를 수와 자료형이 결정되지 않는 체 함수를 호출
- * 출력할 인자의 수와 자료형 인자 `_Format`에 `%d`

•가변인자

- 함수에서 인자의 수와 자료형이 결정되지 않는 함수 인자 방식
- 처음 또는 앞 부분의 매개변수는 정해져 있으나
- 이후 매개변수 수와 각각의 자료형이 고정적이지 않고 변하는 인자
 - * 매개변수에서 중간 이후부터 마지막에 위치한 가변 인자만 가능
- 함수 정의 시 가변인자의 매개변수는 ...으로 기술
- 함수 vatest의 함수 헤드
- void vatest(int n, ...)
 - * 가변 인자인 ...의 앞 부분에서는 반드시 매개변수가 int n 처럼 고정적이어야 한다.
- 가변인자... 시작 전 이전 고정 매개변수
 - * 가변인자를 처리하는데 필요한 정보를 지정하는데 사용한다.

•가변 인자가 있는 함수 구현

- 함수에서 가변 인자를 구현 과정
 - * 필요 매크로함수와 자료형을 위해 헤더파일 stdarg.h가 필요
 - 1. 가변인자 선언
 - * 마치 변수선언처럼 가변인자로 처리할 변수를 하나 만드는 일
 - 2. 가변인자 처리 시작
 - * 선언된 변수에서 마지막 인자를 지정해 가변 인자의 시작 위치를 알리는 방법
 - 3. 가변인자 얻기
 - * 가변인자 각각의 자료형을 지정하여 가변인자를 반환 받는 절차
 - * 매크로 함수 va_arg()의 호출로 반환된 인자로 원하는 연산을 처리한다.
 - 4. 가변인자 처리 종료
 - * 가변 인자에 대한 처리를 끝내는 단계
-
- 가변인자 처리 절차와 가변인자가 있는 함수 sum(int numargs, ...)
 - 가변인자 앞의 첫 고정인자인 numargs는 가변인자의 수
 - int 형인 가변인자를 처리하여 그 결과를 반환하는 함수
 - 가변인자... 시작 전 첫 고정 매개변수
 - * 이후의 가변인자를 처리하는데 필요한 정보를 지정하는데 사용

포인터 전달과 반환

• 매개변수와 반환으로 포인터 사용

주소연산자 &

- 함수에서 매개변수를 포인터로 이용하면 결국 참조에 의한 호출
- 함수원형 `void add(int*,int,int);` 에서 첫 매개변수가 포인터인 `int*`

- * 함수 `add()`는 두 번째와 세 번째 인자를 합해 첫 번째 인자가 가리키는 변수에 저장 함수
- * 변수인 `sum`을 선언하여 주소값인 `&sum`을 인자로 호출

6주차 수업 함수와 포인터 활용

• 주소값 반환

함수의 결과를 포인터로 반환하는 예

- 함수원형을 `int*add(int*, int, int)`로 하는 함수 `add()`
 - * 반환값이 포인터인 `int*`
 - * 두 수의 합을 첫 번째 인자가 가리키는 변수에 저장한 후 포인터인 첫 번째 인자를 그대로 반환
- `add()`를 `*add(&sum,m,n)`호출
 - * 변수 `sum`에 합 `a+b`가 저장
 - * 반환값인 포인터가 가리키는 변수인 `sum`을 바로 참조

• 상수를 위한 `const` 사용

- 포인터를 매개변수로 이용하면 수정된 결과를 받을 수 있어 편리하다.
- 이러한 포인터 인자의 잘못된 수정을 미리 예방하는 방법 즉 수정을 원하지 않는 함수의 인자 앞에 키워드 `const`를 삽입
 - * 참조되는 변수가 수정될 수 없게 한다.
- 키워드 `const`는 인자인 포인터 변수가 가리키는 내용을 수정 불가능

•복소수를 위한 구조체

-구조체 complex

- * 실수부와 허수부를 나타내는 real과 img를 멤버로 구성한다,

-복소수(complex number)

- * 실수의 개념을 확장 수로 $a + bi$ 로 표현
- * 여기서 a와 b는 실수이며, i는 허수단위로 $i^2 = -1$ 을 만족
a는 실수부, b는 허수부
- * 복소수에서의 사칙 연산

- 복소수의 합: $(a + bi) + (c + di) = (a + b) + (c + d)i$
- 복소수의 곱: $(a + bi) * (c + di) = (ac - db) + (ad + bc)i$
- $(a + bi)$ 의 켤레 복소수: $(a - bi)$
- $(a - bi)$ 의 켤레 복소수: $(a + bi)$

•함수 paircomplex ()

-인자인 복소수의 켤레 복소수를 구하여 반환하는 함수

- * 복소수 $(a+bi)$ 의 켤레 복소수는 $(a-bi)$

-구조체는 함수의 인자와 반환값으로 이용이 가능하다.

-다음 함수는 구조체 인자를 값에 의한 호출 방식으로 이용

-함수에서 구조체 지역변수 com을 하나 만들어 실인자의 구조체 값을 모두 복사하는 방식으로 구조체 값을 전달 받음

함수 포인터와 void 포인터

•함수 포인터

-함수 주소 저장 변수

- * 포인터의 장점은 다른 변수를 참조하여 읽거나 쓰는 것도 가능

-함수 포인터

- * 하나의 함수 이름으로 필요에 따라 여러 함수를 사용하면 편리
- * 함수 포인터 pfun은 함수 add()와mult()그리고 subt()로도
사용이 가능하다

-함수 포인터

- * 함수의 주소값을 저장하는 포인터 변수
- * 함수 포인터는 함수를 가리키는 포인터
- * 반환형, 인자목록의 수와 각각의 자료형이 일치하는 함수의 주소를
저장할 수있는 변수

-함수 포인터 선언

- * 함수원형에서 함수이름을 제외한 반환형과 인자목록의 정보가 필요

-변수 이름이 pf인 함수 포인터를 하나 선언

-함수 포인터 pf는 함수 add()의 주소를 저장 가능

- * 함수원형이 void add(double*,double,double)정보 필요
- * 함수원형에서 반환형인 void와 인자목록인 (double*,double,double) 정보 필요

-주의할점

- *(pf)와 같이 변수이름인 pf 앞에는 *이 있어야 하며 반드시 괄호 사용한다.

- * 만일 괄호가 없으면 함수원형

-pf는 함수 포인터 변수가 아니라 void*를 반환하는 함수이름

-함수 포인터 변수 pf

-함수 add()만을 가리킬 수 있는 것이 아니라 add()와 반환형과 인자 목록이 같은 함수는 모두 가리킬 수 있음

-subtract()의 반환형과 인자목록이 add()와 동일하면 pf는 함수 subtract()도 가리킬 수 있다.

•함수 포인터 배열 개념

-원소로 여러 개의 함수 포인터를 선언하는 함수 포인트 배열

-크기가 3인 포인터 배열 pfunary는 문장 int (*pfunary[3]) (int,int); 으로 선언

-배열 pfunary의 각 원소가 가리키는 함수

- * 반환값이 int이고 인자목록이 (int, int)

•함수 포인터 배열 선언

-함수 포인터 배열선언 구문

-배열 fpary의 각 원소가 가리키는 함수

- * 반환값이 void이고 인자목록 (double*,double,double)

-배열 fpary을 선언한 이후에 함수 4개를 각각의 배열원소에 저장

-배열 fpary을 선언하면서 함수 4개의 주소값을 초기화하는 문장

•Void 포인터

-포인터는 주소값을 저장하는 변수

- * int*, double* 처럼 가리키는 대상의 구체적인 자료형의 포인터로 사용이 일반적 이다.
- * 주소값이란 참조를 시작하는 주소에 불과하다.
- * 자료형을 알아야 참조할 범위와 내용을 해석할 방법을 알 수 있다.

-void 포인터 (void*)는 무엇인가?

- * void 포인터는 자료형을 무시하고 주소값만을 다루는 포인터
- * 대상에 상관없이 모든 자료형의 주소를 저장할 수 있는 만능 포인터로 사용이 가능하다.
- * void 포인터에는 일반 포인터는 물론 배열과 구조체 심지어 함수 주소도 저장 가능하다.

•Void 포인터 활용

-void 포인터는 모든 주소를 저장 가능하다.

-가리키는 변수를 참조하거나 수정이 불가능 하다.

-주소값으로 변수를 참조하려면 결국 자료형으로 참조범위를 알아야 하는데 void 포인터는 이러한 정보가 전혀 없이 주소 값을 담는 변수에 불과하기 때문이다.

-void 포인터는 자료형 정보는 없이 임시로 주소 만을 저장하는 포인터

- * 그러므로 실제 void 포인터로 변수를 참조하기 위해서는 자료형 변환이 필요하다.

6주차 수업 파일 처리

파일기초

•파일의 필요성

워드프로세서 없이 프로그램에서 프로그램의 결과로 구성되는 파일을 직접 만들 수 있는가?

-프로그램에서 출력을 파일에 한다면 파일이 생성

-반대로 키보드에서 표준 입력하던 입력을 파일에서 입력하면 파일입력

•파일과 메모리

- 프로그램이 종료되면 모두 사라지는 자료
 - * 변수와 같이 프로그램에서 내부에서 할당되어 사용되는 주기억장치의 메모리 공간
- 프로그램이 종료되더라도 계속 저장
 - * 보조기억장치인 디스크에 저장되는 파일은 직접 삭제하지 않은 한 계속 남는다.
- 프로그램에서 사용하던 정보를 종료 후에도 계속 사용하고 싶다면
 - * 프로그램에서 파일에 그 내용을 저장
 - * 학생 성적 처리를 프로그램을 통하여 결과를 얻어내 그 처리 결과를 지속적으로 저장하려면 파일에 저장

•텍스트 파일과 이진 파일

- 파일:보조기억장치의 정보저장 단위로 자료의 집합
 - * 텍스트 파일과 이진파일 두 가지 유형으로 나뉜다.
- 텍스트 파일: 메모장 같은 편집기로 작성된 파일
 - * 내용이 아스키 코드와 같은 문자 코드값으로 저장
 - * 메모리에 저장된 실수와 정수와 같은 내용도 문자 형식으로 변환되어 저장한다.
 - * 텍스트 편집기를 통하여 그 내용을 볼 수 있고 수정 가능하다
- 이진파일: 실행파일과 그림 파일, 음악파일, 동영상 파일등
 - * 목적에 알맞은 자료가 이진 형태로 저장되는 파일
 - * 컴퓨터 내부 형식으로 저장되는 파일
 - * 목적에 알맞은 자료가 이진 형태로 저장되는 파일
 - * 컴퓨터 내부 형식으로 저장되는 파일
 - * 자료는 메모리 자료 내용에서 어떤 변환도 거치지 않고 그대로 파일에 기록한다
 - * 입출력 속도도 텍스트 파일보다 빠르다.
 - * 메모장과 같은 텍스트 편집기로는 그 내용을 볼 수 없다.
 - * 내용을 이미 알고 있는 특정한 프로그램에 의해 인지될 때 의미가 있다.

•입출력 스트림

- 자료의 입력과 출력은 자료의 이동
 - * 자료가 이동하려면 이동 경로가 필요하다.
- 입출력 스트림
 - * 입출력 시 이동통로
- 표준입력 스트림: 키보드에서 프로그램으로 자료가 이동 경로
 - * 함수 scanf() 표준 입력 스트림에서 자료를 읽을 수 있는 함수
- 표준출력 스트림: 프로그램에서 모니터의 콘솔로 자료가 이동 경로
 - * 함수 printf() 표준출력 스트림으로 자료를 보낼 수 있는 함수
- 입력 스트림: 다른 곳에서 프로그램으로 들어오는 경로
 - * 자료가 떠나는 시작 부분이 자료 원천부
 - * 표준입력: 원천부가 키보드
 - 파일입력:파일이면 파일로부터 자료를 읽는 것
 - 스크린입력: 터치스크린이면 스크린에서 터치정보
 - 네트워크입력: 다른 곳에서 프로그램으로 네트워크를 통해 자료전달
- 출력스트림: 프로그램에서 다른 곳으로 나가는 경로
 - * 자료의 도착 장소 자료 목적부
 - * 표준출력 : 목적부가 콘솔
 - 파일출력: 파일이면 파일에 원하는 값을 저장
 - 프린터출력: 프린터이면 프린터에 출력물
 - 네트워크출력:네트워크이면 네트워크 출력이 되어 다른 곳으로 자료가 이동

•파일 스트림 이해

- 보조기억장치의 파일과 프로그램을 연결하는 전송경로
- 파일 입력 스트림
 - * 파일에서 프로그램으로 자료의 입력을 위한 스트림
- 마찬가지로 파일 출력 스트림
 - * 프로그램에서 파일로 출력을 위한 스트림
- 파일 스트림을 만들기 위해서는 특정한 파일이름과 파일모드가 필요하다.
 - * 여기서 파일 모드란 입력 또는 출력과 같은 스트림의 특징

• 함수 fopen() 파일 스트림 열기

- 함수 fopen() 또는 fopen_s()를 이용 프로그램에서 특정한 파일과 파일 스트림을 연결하는 함수
 - * 헤더 파일 stdio.h 필요하다.
- 현재 Visual C++에서 함수 fopen()는 fopen_s로 대체, 함께 사용 가능하다.
- FILE:헤더 파일 stdio.h에 정의되어 있는 구조체 유형
- 함수 fopen()의 반환값 유형 FILE*은 구조체 FILE의 포인터 유형
- 함수 fopen()은 인자가 파일이름과 파일열기 모드
 - * 파일 스트림 연결에 성공하면 파일 포인터를 반환하며, 실패하면 NULL 을 반환 한다.

• 함수 fopen_s()

- 파일 “basic.tex” 를 여는 모듈
- 파일에 자료를 쓰기 위한 파일 스트림을 연결하기 위해서는 모드 값을 “w” 로 기술한다.
- 함수 fopen_s()는 성공적으로 지정된 외부 파일 이름과 내부 파일 포인터 f와 출력 파일 스트림이 연결되면 FILE 포인터가 인자 f에 저장
 - * 파일 스트림 연결에 성공하면 정수 0을 반환
 - * 만일 스트림 연결에 실패하면 양수를 반환
 - * 첫 번째 인자는 파일 포인터의 주소값
 - * 두 번째 인자인 문자열은 처리하려는 파일 이름
 - * 세 번째 문자열은 파일열기 종류인 모드

파일열기 종류(모드)

- 텍스트 파일인 경우 “r” , “w” , “a” 등의 종류
- 읽기모드 r
 - * 읽기가 가능한 모드이며, 쓰기는 불가능
- 쓰기모드 w
 - * 파일 어디에서든 쓰기가 가능한 모드 이나 읽기는 불가능
- 추가모드
 - * 파일 중간에 쓸 수 없으며 파일 마지막에 추가적으로 쓰는 것만 가능한 모드
 - * 읽기는 불가능 하다.
 - * 파일에 쓰는 내용은 무조건 파일 마지막에 추가

• 함수 fclose()

- fclose()으로 연결한 파일 스트림을 닫는 기능을 수행한다.
- 파일 스트림을 연결한 후 파일 처리가 모두 끝났으면 파일 포인터 f를 인자로 함수 fclose()를 호출하여 반드시 파일을 닫도록 한다.
- 내부적으로 파일 스트림 연결에 할당된 자원을 반납하고, 파일과 메모리 사이에 있던 버퍼의 내용을 모두 지우는 역할을 수행한다.
- 파일 스트림 f를 닫는 함수로서, 성공하면 0을 실패하면 EOF을 반환

텍스트 파일 입출력

• 함수 fprintf()와 fscanf()

- 함수 fprintf()와 fscanf_s()를 이용
- 텍스트 파일에 자료를 쓰거나 읽기 위하여 헤더파일 stdio.h를 포함
 - * 첫 번째 인자는 입출력에 이용될 파일
 - * 두 번째 인자는 입출력에 이용되는 제어 문자열
 - * 다음 인자들은 입출력될 변수 또는 상수 목록
- 함수 fprintf()와 fscanf() 또는 fscanf_s()의 첫번째 인자에 각각 stdin 또는 stdout를 이용하면 표준 입력, 표준 출력으로 이용이 가능하다.
- 기호 상수 stdin, stdout은 stderr
- 헤더 파일 stdio.h에 정의되어 있는 값
- 각각 표준 입력, 표준 출력, 표준에러를 의미한다.

• 함수 fgets()와 fputs()

- 함수 fgets() : 파일로부터 한 행의 문자열을 입력 받는 함수
 - * 차일로부터 문자열을 개행문자(\n)까지 읽어 마지막 개행문자 '\0' 문자로 바꾸어 입력 버퍼 문자열에 저장
 - * 첫 번째 인자는 문자열이 저장될 문자 포인터
 - * 두 번째 인자는 입력할 문자의 최대 수, 세번째 인자는 입력 문자열이 저장될 파일
- 함수 futs() : 파일로 한 행의 문자열을 출력하는 함수
 - * 문자열 한 행에 출력
 - * 첫 번째 인자는 출력될 문자열이 저장된 문자 포인터
 - * 두 번째 인자는 문자열이 출력되는 파일
- fgets()와fputs()는 헤더파일 stdio,h 가 필요하다

• 함수 feof()와ferror()

- 함수 feof(): 파일 스트림이 EOF 표시를 검사하는 함수
 - * 읽기 작업이 파일의 이전 부분을 읽으면 0을 반환하고 그렇지 않으면 0이 아닌 값을 반환한다.
 - * 파일 스트림의 EOF은 이전 읽기 작업에서 EOF 표시에 도달하면 0 0이 아닌 값으로 지정한다.
- 단순히 파일 지시자가 파일의 끝에 있더라도 feof()의 결과는 0이다.
- 함수 ferror() : 파일 처리에서 오류가 발생했는지 검사하는 함수
 - * 이전 파일 처리에서 오류가 발생하면 0이 아닌 값을 반환
 - * 오류가 발생하지 않으면 0을 반환
- 헤더파일 stdio.h 필요.

• 함수 fgetc()와 fputc()

- 파일로부터 문자 하나를 입력받는 함수
- 함수 fputc()와 putc()
 - * 문자 하나를 파일로 출력하는 함수
 - * 함수들은 문자 하나의 입출력의 대상인 파일 포인터를 인자로 사용
- 헤더파일 stdio.h 필요
- getchar()와 putchar()
 - * getc()와 putc()를 이용한 매크로로 정의
 - * getchar()와 putchar()는 함수로도 구현가능

이진 파일 입출력

• 함수 fprintf()와 fsanf_s()

- 자료의 입출력을 텍스트 모드로 처리
- 출력된 텍스트 파일은 텍스트 편집기로 그 내용을 볼 수 있으며 텍스트 파일의 내용은 모두 지정된 아스키 코드와 같은 문자 코드값
 - * 그 내용을 확인할 수 있을 뿐만 아니라 인쇄 가능
- 함수 fprintf()를 이용
 - * int 형 변수 cnt의 값을 파일 f에 출력하는 과정
 - * 실제로 파일에 저장되는 자료는 정수값 10에 해당하는 각 문자의 아스키 값
 - * 각각의 문자 '1' 과 '0' 의 아스키 코드값이 저장

• 함수 fwrite()와 fread()

- 이진파일은 C언어의 자료형을 모두 유지하면서 바이트 단위로 저장되는 파일
- 이진 모드로 블록 단위 입출력을 처리
- 헤더파일 stdio.h 필요하다.
- 함수 fwrite()
 - * 첫 번째 인자 ptr은 출력될 자료의 주소값
 - * 두 번째 인자 size는 출력될 자료 항목의 바이트 크기
 - * 세 번째 인자는 출력될 항목의 개수이며, 마지막 인자는 출력될 파일 포인터
 - * 파일 f에 ptr에서 시작해서 size*n 바이트만큼의 자료를 출력
- 함수 fread()
 - * 이진 파일에 저장되어 있는 자료를 입력
 - * 함수 fwrite()와 인자는 동일

• 함수 fwrite()를 이용 과정

- 바이트 단위로 원하는 블록을 파일에 출력하기 위한 함수
- 출력된 자료는 함수 fread()로 입력해야 그 자료유형을 유지
- 세 번째 항목인 출력 항목 수를 4로 지정 한 경우의 출력
- 이진 파일을 위한 파일 열기 모드
 - * 문자 'b' 를 추가

감사합니다.