

20191776 노지원

C Language

Contents

1

강의계획서

2

1~8주차 강의 정리

단원별로 요약 정리한 내용

3

느낀 점

학생 교과목 포트폴리오를 만들며

1. 강의 계획서

C 언어

2020 학년도 1학기	전공	컴퓨터정보공학과(사회맞춤형 지능형 컴퓨팅과정)	학부	컴퓨터공학부
과 목 명	C예플리케이션구현(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%, 과제를 및 퀴즈 20%, 출석 20%			
수강안내	C 언어를 활용하여 응용프로그램을 구현할 수 있다.			

A large blue geometric shape, resembling a stylized arrow or a corner, pointing towards the right side of the slide.

2. 단원 별 내용 정리

C 언어

11장

[학습목표]

1. 문자와 문자열을 이해할 수 있다.
2. 문자와 문자열 입출력을 이해하고 설명할 수 있다.
3. 문자열 관련 함수를 이해하고 설명할 수 있다.
4. 여러 개의 문자열을 처리하는 방법에 대해 이해하고 설명할 수 있다.

< 학습목표 1 : 문자와 문자열을 이해 >

개념

○ 문자의 개념

- 영어의 알파벳이나 한 글의 한 글자를 작은따옴표로 둘러싸서 'A'와 같이 표기
- 작은따옴표에 의해 표기된 문자를 문자 상수

○ 문자열의 개념

문자의 모임인 일련의 문자를 문자열
문자열은 일련의 문자 앞뒤로 큰 따옴표로 둘러싸서 "java"로 표기
문자의 나열인 문자열은 'ABC'처럼 작은따옴표로 둘러싸도 문자가 될 수 없으며 오류가 발생 -> 반드시 큰 따옴표""

선언

○ 문자의 선언

- char형 변수에 문자를 저장함

```
char ch1 = 'N';  
char ch2 = '가';  
char ch3 = '$';
```

○ 문자열의 선언

문자열을 저장하려면 char형 문자의 모임인 '문자 배열'을 사용함

문자열의 마지막을 의미하는 **NULL문자 '\0'**가 **마지막에 저장**되어야 함

-> IF) NULL문자가 없다면 출력과 같은 문자열 처리에 문제 발생

(즉, 문자열이 저장되는 배열크기는 반드시 저장될 문자 수 보다 1이 커야 함)

배열 크기(배열의 개수) = 문자 수 + 1

문자열 선언 방법 1 <배열 선언 시 초기화>

- 중괄호를 사용
- 문자 하나하나를 쉼표로 구분하여 입력하고 마지막 문자로 NULL 인 '\0'을 삽입

```
char coffee[] = { 'C','O','F','F','E','E','\0' };
```

'C'	'O'	'F'	'F'	'E'	'E'	'\0'
-----	-----	-----	-----	-----	-----	------

문자열 선언 방법 2

<큰 따옴표를 사용해 문자열 상수를 바로 대입>

- 배열 선언 시 저장할 큰 따옴표를 사용해 문자열 상수를 바로 대입하는 방법
- 배열 초기화 시 배열크기는 지정하지 않는 것이 더 편리
- > 자동으로 마지막에 NULL인 '\0'를 채워 줌
- + 선언 없이 `char c[]`처럼 초기 값만 지정하기 (X)

```
char coffee[] = "COFFEE";
```

'C'	'O'	'F'	'F'	'E'	'E'	'\0'
-----	-----	-----	-----	-----	-----	------

문자열 선언 방법 3

<큰 따옴표를 사용해 문자열 상수를 바로 대입 + 배열크기 지정>

- 배열크기 지정 시, 지정한 배열크기가 (문자 수 +1)보다 크면 나머지 부분은 모두 '\0'문자로 채워짐

```
char coffee[10] = "COFFEE";
```

'C'	'O'	'F'	'F'	'E'	'E'	'\0'	'\0'	'\0'	'\0'
-----	-----	-----	-----	-----	-----	------	------	------	------

알게 된 내용

1. 문자는 한 문자 표현, 양쪽에 작은 따옴표를 쓴다.
2. 하나의 문자여도 큰 따옴표로 표시하면 그것은 문자가 아닌 문자열이다.
3. 문자열은 큰 따옴표를 사용하고 문자 배열을 통해 선언한다.
4. 배열 크기(배열의 개수) = 문자 수 + 1
5. 문자열 선언하는 3가지 방법

< 학습목표 2 : 문자와 문자열 입출력을 이해 및 설명 >

○ 함수 printf()를 사용해 문자와 문자열 출력

```
#include <stdio.h>

int main(void) {
    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"; //크기를 생략하는 것이 간편
    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;
}
```

1행 printf()를 쓰기 위해
헤더파일 필요

5행, 20행 %c : 문자를 출력
하는 형식제어문자

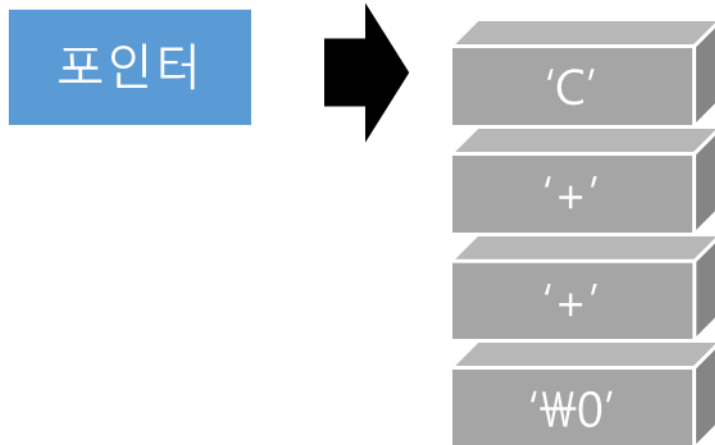
5행 %d : 정수를 출력하는
형식제어 문자

9행, 13행, 17행 %s : 문자열
을 출력하는 형식제어문자

○ 문자열을 구성하는 문자 참조

- 문자열 상수를 문자 포인터에 저장하는 방식
- 문자 하나하나의 수정은 할 수 없음(why? 변수가 첫 번째 메모리 주소를 가리키고 있기 때문)

(+포인터란? : 메모리 공간 주소)



○ 구성하는 문자 참조 출력 방식: **중요**

```
while (*(java + i) != '\0') //java[i]는 *(java + i)와 같음  
    printf("%c", *(java + i++)); //java[i++]은 *(java + i++)와 같음  
printf("\n");
```

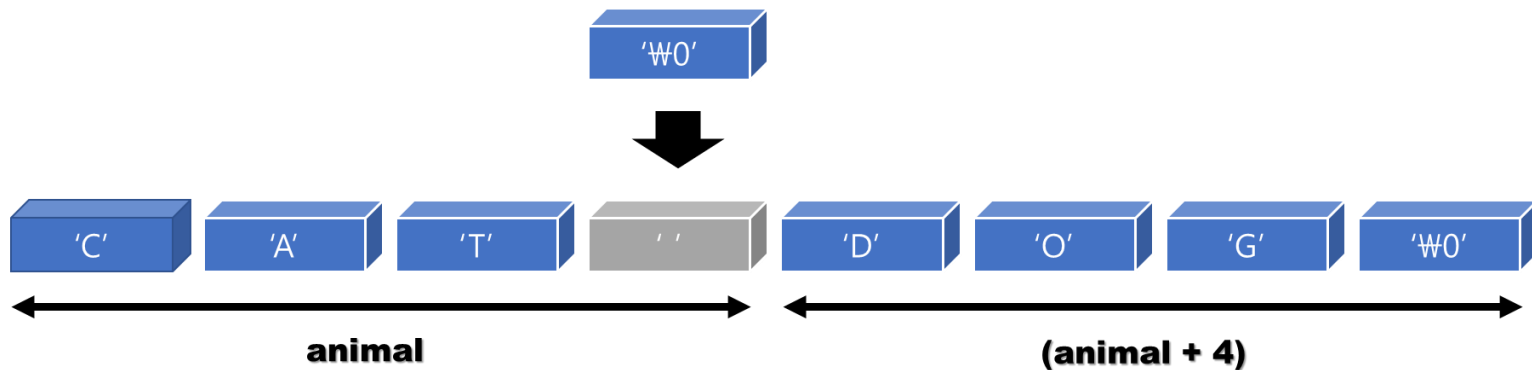
java[i] == *(java + i)

java[i++] == *(java + i++)

○ '\0'문자에 의한 문자열 분리

- 함수 printf()에서 %s는 문자 포인터가 가리키는 위치에서 NULL문자까지 하나의 문자열로 인식함

```
char animal[] = "CAT DOG";  
animal[3] = '\0';  
printf("%s\n%s\n", animal, (animal + 4));
```



○ 다양한 문자 입출력

[입력]

1. 함수 `getchar()` : 라인 버퍼링 방식(버퍼에 저장 후 `enter`를 누르면 버퍼에서 문자를 읽음)

+ 즉각적인 입력을 요구하는 시스템에서는 사용 X

2. 함수 `getche()`, `_getche()` : 버퍼를 사용 하지 않아 문자 하나를 바로바로 입력

3. 함수 `getch()`, `_getch()` : 버퍼를 사용 하지 않고 입력한 문자가 화면에 보이지 않음

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 작동		문자 입력마다 반응	
입력 문자의 표시(echo)	누르면 바로 표시		누르면 바로 표시	표시 안됨
입력문자 수정	가능		불가능	

[출력]

함수 `putchar()`

함수 `_putch()`

○ 문자열 입력

Scanf()

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  int main(void)
5  {
6
7      char name[20], dept[30]; //char *name, *dept; 실행 오류 발생
8
9      printf("%s", "학과 입력 >> ");
10     scanf("%s", dept);
11     printf("%s", "이름 입력 >> ");
12     scanf("%s", name);
13     printf("출력: %10s %10s\n", dept, name);
14
15     return 0;
16 }
```

1행

헤더파일작성 전에

scanf()작동을 위해 씀

gets(), puts()

gets() : 한 행에 문자열을 입력하는 함수

- 마지막에 입력된 '\n'가 '\0'로 교체되어 인자인 배열에 저장

puts() : 한 행에 문자열을 출력하는 함수

- '\0'을 '\n'로 바꿔서 출력

함수 **printf()**와 **scanf()**는 다양한 입출력에 적합하며, 문자열 입출력 함수

puts()와 **gets()**는 처리 속도가 빠르다는 장점이 있음

알게 된 내용

1. `java[i] == *(java + i) / java[i++] == *(java + i++)`
2. `getchar()`, `getche()`, `getch()` 특징
3. `puts()`, `gets()`

< 학습목표 3 : 문자열 관련 함수를 이해하고 설명 >

헤더파일 **string.h**에 함수원형으로 선언된 라이브러리 함수로 제공

1. 문자 배열에 관한 함수

```
#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("문자배열 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;
}
```

<결과 화면>

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

1. **strlen()** : 문자배열의 길이를 반환하는 함수
2. **memcpy()** : 문자배열의 복사를 위한 함수
(null을 포함하여 copy)
3. **memchr()** : 문자 이후의 문자열을 찾는 함수

2. 문자열 비교 함수

〈결과 화면〉

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

```
#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;  
}
```

1. **strcmp()** 인자인 두 문자열 사전 상의 순서로 비교하는 함수(**같으면 0** 이다)

2. **strncmp()** : 비교할 문자의 최대 수를 지정하여 비교하는 함수(**같으면 0** 이다)

- **Jav**와 **java**에서 3개만 비교하기 때문에 같다고 판단 (0)

3. 문자열 복사 함수

```
#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("%d\n", strcpy_s(dest, 80, source));
    //printf("%s\n", dest);
    printf("%s\n", strncpy(dest, "C#", 2));

    printf("%s\n", strncpy(dest, "C#", 3));
    //printf("%d\n", strncpy_s(dest, 80, "C#", 3));
    //printf("%s\n", dest);

    return 0;
}
```

<결과 화면>

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

1. **strcpy()** : 앞 인자 문자열로 뒤 인자 문자열을 복사하는 함수

2. **strncpy()** : 지정된 수 만큼 앞 인자 문자열로 뒤 인자 문자열을 복사하는 함수

- 문자열 마지막 **NULL**까지 포함하여 마지막 줄에는 **C#**만 출력

3. 문자열 연결 함수

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3  #include <string.h>
4
5  int main(void)
6  {
7      char dest[80] = "C";
8
9      printf("%s\n", strcat(dest, " is "));
10     //printf("%d\n", stcat_s(dest, 80, " is "));
11     //printf("%s", dest);
12     printf("%s\n", strncat(dest, "a java", 2));
13     //printf("%d\n", strncat(dsst, 80, "a proce", 2));
14     //printf("%s\n", dest);
15     printf("%s\n", strcat(dest, "procedural "));
16     printf("%s\n", strcat(dest, "language."));
17
18     return 0;
19 }
```

〈결과 화면〉

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

1. **strcat()** : 앞 문자열에 뒤 문자열의 null문자까지 연결하여, 앞의 문자열 주소를 반환하는 함수
 2. **strncat()** : 지정된 수 만큼 앞 문자열에 뒤 문자열 연결하여, 앞의 문자열 주소를 반환하는 함수
- 유의 할 점 : 첫 번째 인자의 넉넉한 공간, 포인터 변수 사용x

4. 문자열 분리 함수★

〈결과 화면〉

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3  #include <string.h>
4
5  int main(void)
6  {
7      char str1[] = "C and C++ language are best!";
8      char *delimiter = ",\t!";
9      //char *next_token;
10
11      printf("문자열 \"%s\"을 >>\n", str1);
12      printf("구분자[%s]를 이용하여 토큰을 추출 >> \n", delimiter);
13      char *ptoken = strtok(str1, delimiter);
14      //char *ptoken = strtok_strtok_s(str, delimiter, &next_token);
15      while (ptoken != NULL)
16      {
17          printf("%s\n", ptoken);
18          ptoken = strtok(NULL, delimiter); //다음 토큰을 반환
19          //ptoken = strtok_s(NULL, delimiter, &next_token); //다음 토큰을 반환
20      }
21
22      return 0;
23
24
25 }
```

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

1. **strtok()** : 문자열에서 구분자(delimiter)인 문자를 여러 개 지정하여 토큰을 추출하는 함수
- 첫 토큰 검사 후 두번째 호출부터는 첫 인자에 **NULL**을 써줘야 다음 토큰을 반환함

5. 문자열 길이와 위치 검색함수

```
#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")); //java의 길이:4
    printf("%s, ", _strlwr(str)); //모두 소문자로 변환
    printf("%s\n", _strupr(str)); //모두 대문자로 변환

    //문자열 VA가 시작되는 포인터 반환: VA 2013 GO C#
    printf("%s, ", strstr(str, "VA"));
    //문자 A가 처음 나타나는 포인터 반환 :AVA 2013 GO C#
    printf("%s\n", strchr(str, 'A'));

    return 0;
}
```

<결과 화면>

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

1. **strlen()** : NULL 문자를 제외한 문자열 길이를 반환하는 함수
2. **strlwr()** : 인자를 모두 소문자로 변환하여 반환
3. **strupr()** : 인자를 모두 대소문자로 변환하여 반환

알게 된 내용

1. 문자 배열에 관련한 함수들
2. 문자열 복사와 연결에 관련된 함수들
3. 문자열 분리에 관한 함수들

< 학습목표 4 : 여러 개의 문자열을 처리하는 방법에 대해 이해하고 설명 >

○ 문자 포인터 배열

- 각각의 문자열 저장을 위한 최적의 공간을 사용
- 문자열 상수의 수정은 불가능

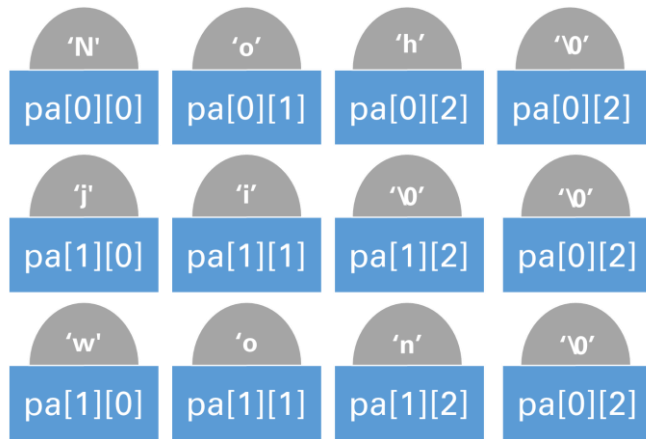


```
char *pa[] = {"Noh", "ji", "won"};
printf("%s ", pa[0]); printf("%s ", pa[1]); printf("%s\n ", pa[2]);
```

○ 이차원 문자 배열

- 문자의 이차원 배열에서 모든 열 수가 동일하게 메모리에 할당 -> 메모리 공간 낭비 !
(But. 요즘은 큰 문제x)
- 문자열 수정 가능
- 두 번째 크기는 문자열에서 가장 긴 문자열의 길이 보다 1크게 지정

```
char ca[][4] = {"Noh", "ji", "won"};  
printf("%s ", ca[0]); printf("%s ", ca[1]);  
printf("%s\n ", ca[2]);
```



○ 명령행 인자

Main(int argc, char *argv[])

argc(명령행에서 입력한 문자열의 수)

argv[](명령행에서 입력한 문자열을 전달 받는 문자 포인터 배열)

- 명령행에서 입력하는 문자열을 프로그램으로 전달하는 방법이 명령행 인자를 사용하는 방법
- 주의할 점은 실행 프로그램 이름도 하나의 명령행 인자에 포함된다는 사실

알게 된 내용

1. 문자 포인터 배열을 이용해 처리 방법(수정 불가능)
2. 이차원 배열을 이용한 문자열 처리 방법(수정 가능, 메모리공간 차지)
3. 명령행에서 입력하는 문자열을 프로그램으로 전달하는 방법

12장

[학습목표]

1. 변수 유효범위를 이해하고 설명할 수 있다.
2. 정적 변수와 레지스터 변수를 이해하고 설명할 수 있다.

< 학습목표 1 : 변수 유효범위 이해 >

유효범위 : 변수의 참조가 유효한 범위(아래의 크게 두 가지로 나뉨)

지역 변수

- 유효범위 - 함수, 블록
- 특징
 - 선언 후 초기화하지 않으면 쓰레기 값이 저장되므로 주의해야 함
 - 지역 변수가 할당되는 메모리 영역을 스택(stack)
 - 선언된 부분에서 자동으로 생성되고 함수나 블록이 종료되는 순간 메모리에서 자동으로 제거 : 자동변수(auto)
- > 생략 가능

전역 변수

- 유효범위 - 프로젝트, 파일
- 특징
 - 선언 되면 자동으로 초기화
 - 전역변수와 같은 이름으로 지역변수 선언 가능(but, 사용 권장x)
 - 다른 파일에서 참조 가능하지만 꼭!! Extern을 사용해야 함

<변수의 유효범위 참고 그림>

```
Int global; //전체 전역
Int main(void)
{

}
```

```
void sub()
{
  Int local; //지역변수
}
```

```
Extern int global; //전역 변수를 다른 파일에 사용하기 위함
```

```
Int main(void)
{

}
```

```
Static int staticvar;
//파일 전역
```

```
Int main(void)
{

}
```

○ 전역 변수 장단점

- 전역변수의 선언 위치가 참조하려는 위치보다 뒤에 있는 경우 : `extern` 을 사용해야 함(같은 파일이라도) ->권장 x
- 장점 : 어디서든 수정 가능
- 단점 : 예상하지 못한 값이 저장될 수도 있음

→결론적으로 전역변수는 특별한 경우 사용을 권고 함

< 학습목표 2 : 정적 변수와 레지스터 변수를 이해 >

○ 기억부류(storage class) 개념 및 종류

: 할당되는 메모리 영역이 결정되고 할당과 제거 시기가 결정 해주는 역할

-> auto, register, static, extern : extern제외하고 나머지는 초기화 값 저장 가능

○ 기억부류 (storage class) 유효범위

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

○ register

- 변수의 저장 공간이 일반 메모리가 아닌 **CPU 내부의 레지스터**에 할당되는 변수
- 키워드 register은 자료형 앞에 선언 ex) register int num;
- 일반 메모리에 할당되는 변수가 아니므로 **주소 연산자 &를 사용할 수 없음**
- 처리 **속도를 증가** 시키려는 변수에 이용
- 초기화 값 지정x -> 쓰레기 값 저장 됨



○ 정적변수(static)

- 정적 지역변수, 정적 전역변수 크게 두가지로 나뉨
- 초기값을 저장하지 않아도 자동으로 NULL or '\0' 이 저장 됨
- 초기화는 반드시 상수로만 가능

정적 전역변수(static)

참조 범위는 선언된 파일에만 한정되며 변수의 할당과 제거는 전역 변수 특징을 가짐

extern에 의해 다른 파일에서 참조 불가능

정적 지역변수(static)

참조 범위는 지역변수이면서 변수의 할당과 제거는 전역 변수 특징을 가짐

○ 메모리 영역

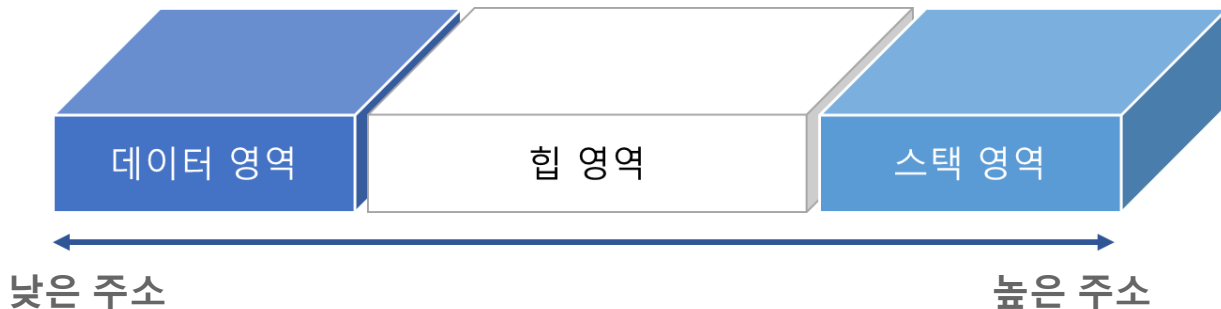
- 메인 메모리의 영역은 프로그램 실행 과정에서

데이터(data) 영역, 힙(heap) 영역, 스택(stack) 영역 세부분으로 나뉨

*데이터 영역: 전역변수와 정적변수가 할당되는 저장 공간 / 낮은 주소에서 높은 주소로 할당 -> 고정된 메모리 영역 확보

*힙 영역: 동적 할당되는 변수가 할당되는 저장공간 / 데이터 영역과 스택 영역 사이에 위치

*스택 영역: 함수 호출에 의한 형식 매개변수 그리고 함수 내부의 지역 변수가 할당되는 저장공간 / 높은 주소에서 낮은 주소로 할당
->함수의 호출에 따라 할당과 제거 반복



알게 된 내용

1. 실행 속도를 개선 -> 지역변수인 레지스터 변수 이용
2. 함수나 블록이 종료 되더라도 계속적으로 값을 저장 -> 정적 지역변수
3. 해당 파일 내부에서만 변수를 공유 -> 정적 전역 변수
4. 변수의 종류에 따른 생존기간과 유효 범위

13장

[학습목표]

1. 구조체와 공용체를 이해하고 설명할 수 있다.
2. 자료형 재정의에 위한 typedef를 사용할 수 있다.
3. 구조체 포인터와 배열을 활용할 수 있다.

< 학습목표 1 : 구조체와 공용체를 이해 >

○ 구조체의 개념과 정의 방법

- 구조체의 개념 : 정수나 문자, 실수나 포인터 그리고 이들의 배열 등을 묶어 하나의 자료형으로 이용한 것

+ 유도 자료형 : 기존 자료형으로 새로이 만들어진 자료형

- 구조체의 정의 : 구조체를 자료형으로 사용하려면 구조체 정의 필수!

-> 구조체 틀 정의 : 각 구조체 멤버에 초기값x, 다른 구조체 변수 및 구조체 포인터도 허용

```
struct person
{
    char name[20]; //이름
    int age; //나이
    char s[10]; //성별
};
```



```
struct person nohji;
```

○ 구조체 변수 선언과 초기화

- 구조체 변수 선언

```
struct 구조체태그이름 변수명;
```

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

→ 구조체 정의와 변수 선언 함께

- 구조체 변수의 초기화

: 중괄호 내부에서 구조체의 각 멤버 정의 순서대로 초기값을 쉼표로 구분하여 기술
(기술되지 x → 기본값 0,0.0,'\0'등으로 저장)

```
Struct account mine = {"노지원",1001,400000};
```

○ 구조체의 멤버 접근 연산자 . 와 변수 크기


- 선언된 구조체형 변수는 접근연산자 .를 사용하여 멤버를 참조 할 수 있음
- 실제 구조체의 크기는 멤버의 크기의 합보다 크거나 같음

○ 구조체의 활용

- 구조체 멤버로 사용되는 구조체

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

```
struct person
{
    struct date born;
    char name[12];
    char s[10];
};
```



- 구조체 변수의 대입과 동등 비교 : 대입은 가능하지만 동등비교를 불가능(멤버끼리 비교 해야 함)

```
struct date date1 = { 1999,10,14 };
Struct date date2 ;
date2 = date1; //대입
```

→ 대입 가능

```
If(date1 == date2)
printf("내용이 같은 구조체"); //오류
```



```
If(date1.day ==date2.day)
printf("day내용이 같은 구조체"); //비교 가능
```

○ 공용체의 활용

- 공용체 개념 : 동일한 저장 장소에 여러 자료형을 저장하는 방법
- 공용체 정의와 변수 선언 → **union** 을 사용 (그 외 struct과 동일)
- 공용체 변수의 크기는 멤버 중 가장 큰 자료형의 크기로 정해 짐)
- 공용체의 초기화 값은 공용체 정의 시 처음 선언한 멤버의 초기값으로만 저장 가능
- 공용체의 멤버는 모든 멤버가 동일한 저장 공간을 사용하므로 동시에 여러 멤버의 값을 동시에 저장하여 이용할 수 없으며 마지막에 저장된 단 하나의 자료 값만 저장함
- 공용체의 멤버 접근 : 구조체와 같이 접근연산자 .를 사용

< 학습목표 2 : 자료형 재정의를 위한 typedef를 사용 >

○ 자료형 재정의 typedef

- typedef 구문

: 이미 사용되는 자료 유형을 다른 새로운 자료형 이름으로 재정의할 수 있도록 하는 키워드

: 재정의 이유는 프로그램의 시스템 간의 호환성과 편의성을 위함

typedef 기존자료유형 새로운자료형1, 새로운자료형2;

typedef int inter, interger;

inter age1; // int age1 동일
interger age2; //int age2 동일

○ 구조체 자료형 재정의

- struct를 생략한 새로운 자료형

```
typedef struct
{
    char title[30];
    char company[30];
    char kinds[30];
} software;
```

→ software는 변수가 아니라 새로운 자료형

< 학습목표 3 : 구조체 포인터와 배열을 활용 >

○ 구조체 포인터

- 포인터 변수 선언 : 각각의 자료형 저장 공간의 주소를 저장하듯이 구조체 포인터는 구조체의 주소 값을 저장할 수 있는 변수

```
struct person
{
    char name[20]; //이름
    int age; //나이
    char s[10]; //성별
};
typedef struct person person;
person hero = {"슈퍼맨", 33, "남"};
Person *p = &hero;
```



○ 포인터 변수의 구조체 멤버 접근 연산자 →

: 연산식 $p \rightarrow \text{name}$ 은 포인터 p 가 가리키는 구조체 변수의 멤버 name 을 접근 함

: 연산식 $*p.\text{name}$ 은 접근연산자(.)가 간접연산자(*)보다 우선순위가 빠르므로 $*(p.\text{name})$ 과 같은 연산식

```
1  #include <stdio.h>
2
3  struct lecture
4  {
5      char name[20]; //강좌명
6      int type; //강좌구분 0: 교양, 1: 일반선택, 2: 전공필수, 3: 전공선택
7      int credit; //학점
8      int hours; //사수
9  };
10 typedef struct lecture lecture;
11
12 //제목을 위한 문자열
13 char *head[] = { "강좌명", "강좌구분", "학점", "사수" };
14 //강좌 종류를 위한 문자열
15 char *lectype[] = { "교양", "일반선택", "전공필수", "전공선택" };
16
17 int main(void)
18 {
19     lecture os = { "운영체제", 2, 3, 3 };
20     lecture c = { "C프로그래밍", 3, 3, 4 };
21     lecture *p = &os;
22
23     printf("구조체크기: %d, 포인터크기: %d\n", sizeof(os), sizeof(p));
24     printf("%10s %12s %6s %6s\n", head[0], head[1], head[2], head[3]);
25     printf("%12s %10s %5d %5d\n", p->name, lectype[p->type], p->credit, p->hours);
26
27     //포인터 변경
28     p = &c;
29     printf("%12s %10s %5d %5d\n", (*p).name, lectype[(*p).type], (*p).credit, (*p).hours);
30     printf("%12s %10s %5d %5d\n", *c.name, lectype[c.type], c.credit, c.hours);
31
32     return 0;
33 }
```

25행 강좌구분 변수 type 에 저장된 값으로 문자 포인터 배열의 첨자를 사용

30행 $*c.\text{name}$ 은 $*(c.\text{name})$ 과 같아 C만 출력

○ 구조체 배열

- 다른 배열과 같이 동일한 구조체 변수가 여러 개 필요하면 구조체 배열을 선언하여 이용할 수 있음
- 배열 크기 : `sizeof(a)/sizeof(a[0])` or `sizeof(a)/sizeof(lecture)`

```
lecture a[] = {{“인간과사회”,0,2,2},  
               {“취업성공전략”,1,3,3},  
               {“C언어”,2,3,3}};
```

- 배열 주소 저장

```
lecture *p = a;  
For( i=0; i<arysize; i++)  
    printf(“%16s %10s %5d %5d\n”,p[i].name,  
    lecture[p[i].type], p[i].credit,p[i].hours);
```


알게 된 내용

1. 구조체, 공용체의 선언과 초기화
2. 구조체 포인터와 배열
3. 포인터 변수의 구조체 멤버 접근 연산자 →

14장

[학습목표]

1. 함수의 인자전달 방식을 이해하고 설명할 수 있다.
2. 함수에서 인자로 포인터의 전달과 반환으로 포인터 형의 사용을 이해하고 설명할 수 있다.
3. 함수 포인터를 이해하고 설명할 수 있다.

< 학습목표 1 : 함수의 인자전달 방식 이해 >

○ 값에 의한 호출과 참조에 의한 호출

- 함수에서 값의 전달 : c언어는 함수의 인자 전달 방식이 기본적으로 값에 의한 호출 방식.
- + 값에 의한 호출 방식이란? 호출 시 실인자의 값이 형식인자에 복사되어 저장된다는 의미
(call by value)

! 값에 의한 호출 방식을 사용해서는 함수 외부의 변수를 함수 내부에서 수정할 수 없는 특징

```
Int main(void)
{
    int num=10;
    increase(num,20);
    printf("%d",num);
}
```

10

num

```
Void increase(int
origin, int increment)
{
    origin += increment;
}
```

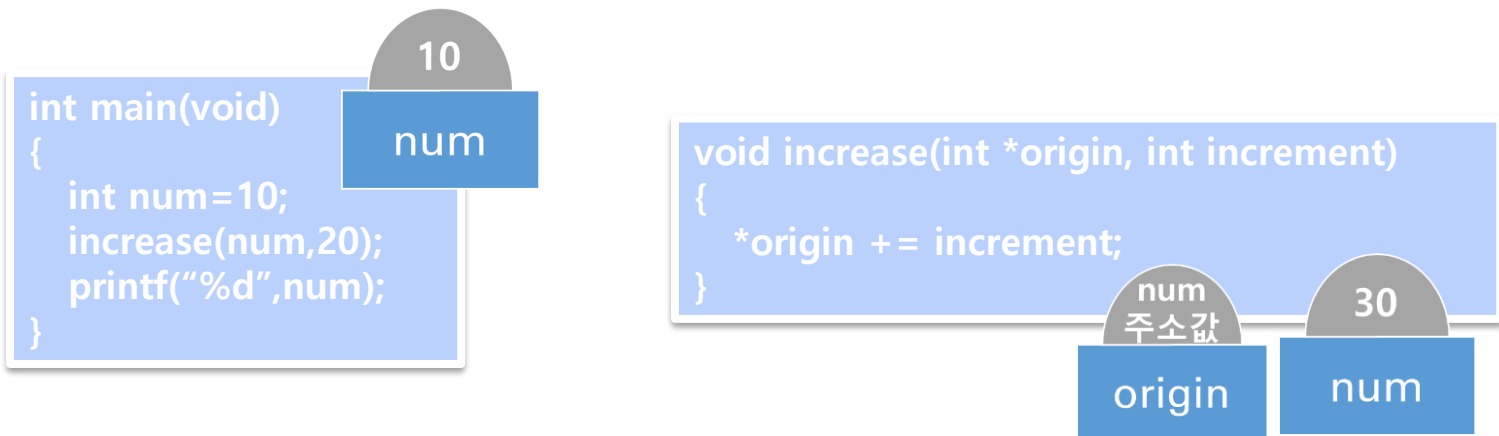
30

origin

- 함수에서 주소의 전달 : c언어에서 포인터를 매개변수로 사용하면 함수로 전달된 실인자의 주소를 이용하여 그 변수를 참조 함

(call by reference)

! C언어에서 포인터를 매개변수로 사용하면 함수로 전달된 실인자의 주소를 이용하여 그 변수를 참조 가능



○ 배열의 전달

- 배열이름으로 전달

: 함수의 매개변수로 배열을 전달하는 것은 배열의 첫 원소를 참조 매개변수로 전달하는 것과 동일

```
1 #include <stdio.h>
2
3 #define ARYSIZE 5
4 double sum(double g[], int n); //배열원소 값을 모두 더하는 함수원형
5
6 int main(void)
7 {
8     //배열 초기화
9     double data[] = { 2.3,3.4,4.5,6.7,9.2 };
10
11     //배열원소 출력
12     for (int i = 0; i < ARYSIZE; i++)
13         printf("%5.1f", data[i]);
14     puts("");
15
16     //배열 원소값을 모두 더하는 함수호출
17     printf("합: %5.1f\n", sum(data, ARYSIZE));
18
19     return 0;
20 }
21
22 //배열 원소값을 모두 더하는 함수정의
23 double sum(double ary[], int n)
24 {
25     double total = 0.0;
26     for (int i = 0; i < n; i++)
27         total += ary[i];
28
29     return total;
30 }
```

4행 함수원형으로 `double sum(double *,int n);`
가능 - 함수원형에서 이름 배열 이름 생략 가능

17행 함수의 배열 매개변수에서 실인자를
호출할 때는 배열이름 `data`를 사용하므로,
`sum(data, ARYSIZE)`로 호출

- 다양한 배열원소 참조 방법

(간접연산자*를 사용한 배열원소의 참조 방법)

```
int i, sum =0; int point[]={12,14,15,6};  
int *address = point;  
int aryLength =sizeof(point)/sizeof(int);
```

```
for( i=0; aryLength, i++)  
    sum+=*(point+i);
```

```
for( i=0; aryLength, i++)  
    sum+=*(address++);
```

```
for( i=0; aryLength, i++)  
    sum+=*(point++);
```

(함수헤더의 배열 인자와 함수정의에서 다양한 배열원소의 참조 방법)

```
For(i=0; i<SIZE; i++)  
{  
    sum+= *ary++;  
}
```

```
For(i=0; i<SIZE; i++)  
{  
    sum+= *(ary++);  
}
```

→ ary는 주소 값을 저장하는 변수이므로 증가연산자 이용 가능 / 후위 증가연산자의 우선순위가 가장 높기에 두개의 의미는 동일함

- 배열크기 계산 방법

$(\text{sizeof}(\text{배열이름}) / \text{sizeof}(\text{배열원소})) \rightarrow \text{배열크기}$

- 다차원 배열 전달

: 다차원 배열을 인자로 이용하는 경우, 함수원형과 함수정의의 헤더에서 첫 번째 대괄호 내부의 크기를 제외한 다른 모든 크기는 반드시 기술

```
double sum(double[][3], int, int);
```

```
void sum (double[][3], int rowsize, int colsize);
```

X[0][0]	X[0][1]	X[0][2]
X[1][0]	X[1][1]	X[1][2]
X[2][0]	X[2][1]	X[2][2]
X[3][0]	X[3][1]	X[3][2]



배열의 전체 원소 수 :12
 $\text{sizeof}(x) / \text{sizeof}(x[0][0])$

배열의 행 수: 4
 $\text{sizeof}(x) / \text{sizeof}(x[0])$

배열의 열 :3
 $\text{sizeof}(x[0]) / \text{sizeof}(x[0][0])$

○ 가변인자

:함수에서 인자의 수와 자료형이 결정되지 않은 함수 인자 방식

- 가변 인자가 있는 함수머리

: 출력할 인자의 수와 자료형은 인자 `_Format`에 `%d` 등으로 표현

```
//함수 printf()의 함수원형  
int printf(const char *_Format, ...);
```

```
//함수 사용 예  
Printf("%d %d %f", 3,4,5.67);  
Printf("%d %d %f %f %f", 3,4,5.67,7,89,4,67);
```

!!주의!!

`double vafun1(char *type,...,int n);` //마지막 고정x

`double vafun2(...);` //처음부터 가변인자일 수 없음

- 가변 인자가 있는 함수구현

: 가변인자 선언, 가변인자 처리 시작, 가변인자 얻기, 가변인자 처리 종료 4단계 필요

: 헤더파일 stdarg.h 필요

1. 가변인자 선언

va_list 가변인자변수; → va_list argp;

2. 가변인자 처리 시작

va_start(가변인자변수, 가변인자_이전_첫_고정인자) → va_start(va_list argp, prevarg)

3. 가변인자 얻기

va_arg(가변인자변수, 반환될_자료형) → type va_arg(va_list argp, type)

4. 가변인자 처리 종료

va_end(가변인자변수) → va_end(va_list argp)

◀ 학습목표 2 : 함수에서 인자로 포인터의 전달과 반환으로 포인터 형의 사용을 이해 ▶

○ 매개변수와 반환으로 포인터 사용

- 주소연산자 & : 함수에서 매개변수를 포인터로 이용하면 결국 참조에 의한 호출

```
Int m=0, n=0, sum=0;  
scanf("%d %d", &m, &n);  
add(&sum, m, n);
```

```
void add(int *sum, int a, int b)  
{  
    *sum = a + b;  
}
```

- 주소값 반환 : 함수결과를 포인터로 반환

```
Int * add(int *, int, int);  
Int m =0, n=0, sum=0;  
scanf("%d %d", &m, &n);  
Printf("두 정수 합: %d\n", *add(&sum, m, n));
```

```
Int * add(int *psum, int a, int  
b)  
{  
    *psum = a + b;  
    return psum;  
}
```

! 지역변수 주소 값의 반환은 문제를 발생 → 함수가 종료되는 시점에 메모리에서 제거되기 때문

○ 상수를 위한 **const** 사용

- 키워드 **const** : 수정을 원하지 않는 함수의 인자 앞에 키워드 **const**를 삽입

```
void multiply(double *result, const double *a, const double *b)
{
    *result = *a * *b;
    *a = *a + 1; //오류
    *b = *b + 1; //오류
}
```

○ 함수의 구조체 전달과 반환

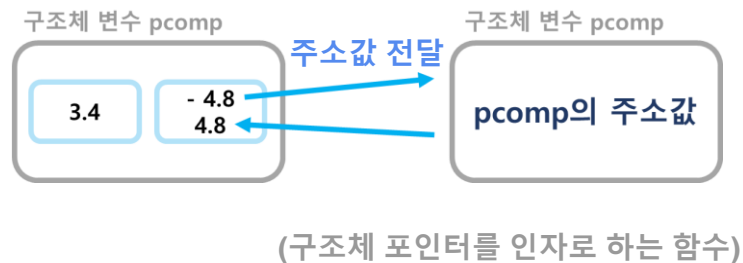
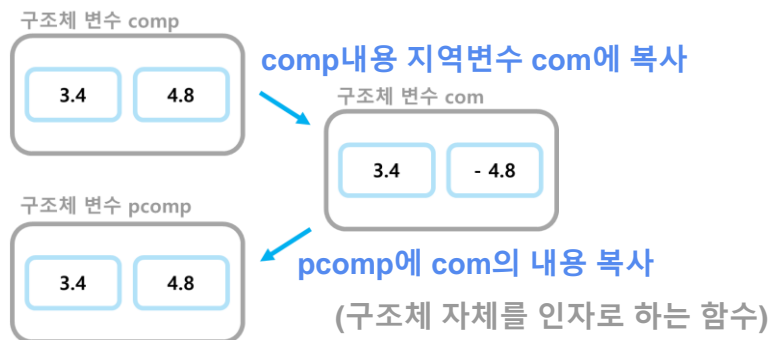
- 복소수를 위한 구조체: 구조체 **complex**를 이용하여 복소수 연산에 이용되는 함수

```
struct complex
{
    double real; //실수
    double img; //허수
};
typedef struct complex complex;
```

- 인자와 반환형으로 구조체 사용

: 함수 `paircomplex1()`는 인자인 복소수의 켤레 복소수를 구하여 반환하는 함수

```
11 void printcomplex(complex com);
12 complex paircomplex1(complex com);
13 void paircomplex2(complex *com);
14
15 int main(void)
16 {
17     complex comp = { 3.4, 4.8 };
18     complex pcomp;
19
20     printcomplex(comp);
21     pcomp = paircomplex1(comp);
22     printcomplex(pcomp);
23     paircomplex2(&pcomp);
24     printcomplex(pcomp);
25     return 0;
26 }
27
28 //구조체 자체를 인자로 사용
29 void printcomplex(complex com)
30 {
31     printf("복소수(a + bi) = %5.1f + %5.1f %n", com.real, com.img);
32 }
33 //구조체 자체를 인자로 사용하여 처리된 구조체를 다시 반환
34 complex paircomplex1(complex com)
35 {
36     com.img = -com.img;
37     return com;
38 }
39 //구조체 포인터를 인자로 사용
40 void paircomplex2(complex *com)
41 {
42     com->img = -com->img;
```



< 학습목표 3 : 포인터 함수의 이해 >

○ 함수 포인터

: 함수의 주소값을 저장하는 포인터 변수

- 함수 주소 저장 변수

: 포인터의 장점은 다른 변수를 참조하여 읽거나 쓰는 것도 가능

→ 하나의 이름으로 필요에 따라 여러 함수를 사용

```
void (*pf)(double *, double, double); //괄호 필수!  
pf = add; //pf=add()는 오류 발생  
pf = mult; //pf=mult()는 오류 발생
```

- 함수 포인터를 이용한 함수 호출

```
double m, n, result = 0;  
void (*pf)(double*, double, double);  
pf=add;  
pf(&result, m, n); //add(&result, m, n);
```

```
void add(double *z, double x, double y)  
{  
    *z = x + y;  
}
```

○ 함수 포인터 배열

- 함수 포인터 배열 개념 : 함수 포인터가 원소인 배열

```
반환자료형 (*배열이름[배열크기])(자료형1 매개변수이름1, 자료형2 매개변수이름2,...);
```

```
반환자료형 (*배열이름[배열크기])(자료형1, 자료형2,...); //매개변수이름 생략 가능
```

(EX)

```
void add(double*, double, double);  
void subtract(double*, double, double);  
void multiply(double*, double, double);  
void device(double*, double, double);  
...  
void(*fpary[4])(double*, double, double);  
fpary[0] = add; fpary[1] = subtract; fpary[2] = multiply; fpary[3] = device;
```

함수 포인터 배열 선언과 초기화 동시에

```
void(*fpary[4])(double*, double, double) = {add, subtract, multiply, device};
```

○ void 포인터 배열

- void 포인터 개념

: 자료형을 무시하고 주소값만을 다루는 포인터 (만능 포인터)

: 일반 변수 포인터, 배열, 구조체, 함수 주소를 담을 수 있음

```
char ch = 'A'; int data = 5; double value = 34.56;  
void *vp ;  
vp = &ch; vp = &data; vp = &value;
```

- void 포인터 활용

: 모든 주소를 저장할 수 있지만 가리키는 변수를 참조하거나 수정이 불가능

→ void 포인터 변수를 참조하기 위해서는 자료형 변환이 필요

```
int m = 10;  
void *p = &m;  
int n = *(int *)p;    //int *로 변환  
n = *p //오류 발생
```

알게 된 내용

1. 함수의 인자전달 방식 (call by value, call by reference)
2. 매개변수와 반환으로 포인터 사용
3. 함수 포인터 , void포인터

15장

[학습목표]

1. 텍스트 파일과 이진 파일의 차이를 이해하고 설명할 수 있다.
2. 텍스트 파일의 입출력 함수를 이해하고 설명할 수 있다.
3. 이진 파일의 입출력 함수를 이해하고 설명할 수 있다.

< 학습목표 1 : 텍스트 파일과 이진 파일의 차이를 이해 >

○ 텍스트 파일과 이진 파일

- 파일의 필요성 : 보조기억장치인 디스크에 저장되는 파일은 직접 삭제하지 않은 한 프로그램이 종료되더라도 계속 저장 가능

- 텍스트 파일과 이진 파일

텍스트 파일 : 문자 기반의 파일로서 내용이 아스키코드와 같은 문자 코드 값으로 저장

이진 파일

: 그림 파일, 동영상 파일, 실행 파일과 같이 각각의 목적에 알맞은 자료가 이진 형태로 저장되는 파일

: 컴퓨터 내부 형식으로 저장되는 파일

: 자료는 메모리 자료 내용에서 어떤 변환도 거치지 않고 그대로 파일에 기록

→ 입출력 속도가 텍스트 파일에 비해 빠름

-입출력 스트림 : 자료의 이동 통로(자료의 이동: 자료의 입력과 출력)

키보드에서 프로그램으로 자료가 이동하는 경로 → 표준입력 스트림

함수 scanf() : 표준입력 스트림에서 자료를 읽을 수 있는 함수

프로그램에서 모니터의 콘솔로 자료를 이동하는 경로 → 표준출력 스트림

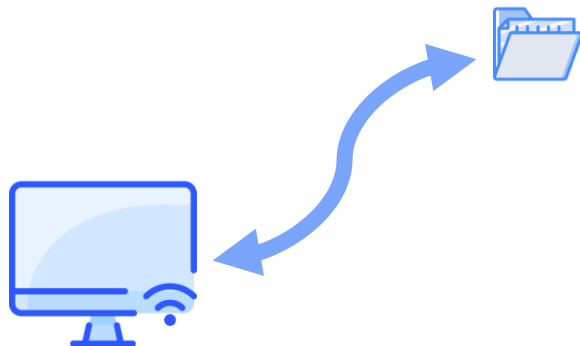
함수 printf() : 표준출력 스트림으로 자료를 보낼 수 있는 함수

- 파일 스트림 이해

: 보조기억장치의 파일과 프로그램을 연결하는 전송 경로

: 프로그램에서 보조기억장치에 파일로 정보를 저장하거나 파일에서 정보를 참조하려면
파일에 대한 파일 스트림을 먼저 연결

: 파일 입력 스트림 / 파일 출력 스트림으로 나뉨



○ 파일 스트림 열기

- 함수 fopen()으로 파일 스트림 열기(fopen() or fopen_s())

```
FILE * fopen(const char * _Filename, const char * _Mode)  
Errono_t fopen_s(FILE ** _File, const * _Filename, const char * _Mode);
```

fopen()은 _Filename의 파일 스트림을 모드_Mode로 연결하는 함수 / 성공 시 파일 포인터를 반환, 실패 시 NULL를 반환
fopen_s()는 성공 시_File에 파일 포인터가 저장되고 정수 0을 반환, 실패 시 양수를 반환

파일 열기 종류(모드)에는 텍스트 파일인 경우 "r", "w", "a" 등이 있음

r : 읽기 모드이며 쓰기는 불가능

w : 쓰기 모드이며 읽기는 불가능

a : 파일 중간에 쓸 수 없으며 파일 마지막에 추가적으로 쓰는 것만 가능한 모드, 읽기는 불가능

- 함수 fclose()로 파일 스트림 닫기

: 파일 스트림을 연결한 후 파일 처리가 모두 끝났으면 파일 포인터 f를 인자로 함수 fclose()를 호출하여 반드시 파일을 닫도록 함

```
int fclose(FILE * _File);  
fclose(f);
```

< 학습목표 2 : 텍스트 파일 입출력 이해 >

○ 파일에 서식화된 문자열 입출력

- 함수 fprintf() / 함수 fscanf()

```
Int fprintf(FILE * _File, const char * _Format, ...);  
Int fscanf(FILE * _File, const char * _Format, ...);  
Int fscanf_s(FILE * _File, const char * _Format, ...);
```

(표준 파일의 종류(첫번째 인자))

표준 파일	키워드	장치
표준 입력	stdin	키보드
표준 출력	stdout	모니터 화면
표준 에러	stderr	모니터 화면

○ 파일문자열 입출력

-함수 fgets() / 함수 fputs()

```
char * fgets(char * _Buf, int _MaxCount, FILE * _File);  
int fputs(char * _Buf, FILE * _File);
```

fgets()는 _File로부터 한 행의 문자열을 _MaxCount 수의 _Buf 문자열에 입력 수행
fputs()는 _Buf 문자열을 _File에 출력 수행

-함수 feof() / 함수 ferror()

: feof()은 파일 스트림의 EOF 표시를 검사하는 함수

: ferror()는 파일 처리에서 오류가 발생했는지 검사하는 함수

```
Int feof(FILE * _File);  
Int ferror(FILE * _File);
```

feof()은 _File의 EOF를 검사

ferror()는 _File에서 오류발생 유무를 검사

○ 파일문자 입출력

-함수 fgetc() / 함수 fputc()

: fgetc()와 getc()는 파일로부터 문자 하나를 입력 받는 함수

: fputc()와 putc()는 문자 하나를 파일로 출력하는 함수

```
Int fgetc(FILE * _File);  
Int fputc(int _CH, FILE * _File);
```

```
Int getc(FILE * _File);  
Int putc(int _CH, FILE * _File);
```

fgetc()와 getc()는 _File에서 문자 하나를 입력 받는 함수
fputc()와 putc()는 _Ch를 파일 _File에 출력하는 함수

콘솔의 표준입출력 함수로는 _getche()와 _putch()를 이용
비주얼 C++에서는 getche(), putch() 권장 함수

< 학습목표 3 : 이진 파일 입출력 이해 >

○ 텍스트와 이진 파일 입력과 출력

- 함수 `fprintf()` / 함수 `fscanf_s()` : 자료의 입출력을 텍스트 모드로 처리
- 함수 `write()`와 `fread()` : C언어의 자료형을 모두 유지하면서 바이트 단위로 저장되는 파일이므로 이진 모드 블록 단위 입출력을 처리

```
size_t fwrite(const void *ptr, size_t size, size_t n, FILE *f);  
size_t fread(void *dstbuf, size_t size, size_t n, FILE *f);
```

`Fwrite()`는 `ptr`이 가리키는 메모리에서 `size`만큼 `n`개를 파일 `f`에 쓰는 함수
`Fread()`는 반대로 파일 `f`에서 `size`의 `n`개 만큼 메모리 `dstbuf`에 읽어오는 함수,
반환값은 성공적으로 입출력을 수행한 항목의 수

○ 구조체의 파일 입출력

```
6 struct personscore //구조체 struct personscore 정의
7 {
8     int number;
9     char name[40];
10    int mid;
11    int final;
12    int quiz;
13 };
14 typedef struct personscore pscore;
15
16 int main()
17 {
18     char fname[] = "score.bin";
19     FILE *f;
20     //쓰기 모드로 파일 열기
21     if (fopen_s(&f, fname, "wb") != 0)
22     {
23         //if (fopen_s(&f, fname, "wb")) == NULL
24         printf("파일이 열리지 않습니다.\n");
25         exit(1);
26     };
27     //표준 입력으로 행을 저장하기 위한 변수
28     char line[80];
29     int cnt = 0; // 입력 학생 번호(자동 생성) 변수
30     pscore score; //구조체 변수 선언
31     printf("이름과 성적(중간, 기말, 퀴즈)을 입력하세요.\n");
32
33     fgets(line, 80, stdin);
34     while (!feof(stdin))
35     {
36         //표준입력의 한줄을 구조체의 멤버 별로 자료를 입력
37         //sscanf(line, "%s %d %d %d", score.name, &score.mid, &score.final, &score.quiz);
38         sscanf_s(line, "%s %d %d %d", score.name, 40, &score.mid, &score.final, &score.quiz);
39         score.number = ++cnt;
40         fwrite(&score, sizeof(pscore), 1, f);
41         fgets(line, 80, stdin);
42     }
43     fclose(f);
44
45     return 0;
46 }
```

6-13행 구조체 struct personscore 정의

14행 구조체 struct personscore 자료형
pscore로 정의

40행 구조체 변수 score를 파일 f에 이진
파일로 저장

41행 다시 표준입력으로 받은 한 행을 변수
line에 저장

43행 파일 처리가 종료되었으면 fclose()로
스트림을 닫음

○ 파일에서 학생 성적 구조체 읽기

```
6 struct personscore
7 {
8     int number;
9     char name[40];
10    int mid;
11    int final;
12    int quiz;
13 };
14 typedef struct personscore pscore;
15 void printhead();
16 int main() {
17     char fname[] = "score.bin";
18     FILE *f;
19
20     if ((f = fopen(fname, "rb")) == NULL){
21         printf("파일이 열리지 않습니다.\n");
22         exit(1);
23     };
24     printhead();
25     //이진모드로 파일 f에서 구조체 pscore 자료 읽기
26     pscore score;
27     fread(&score, sizeof(pscore), 1, f);
28     while (!feof(f))
29     {
30         //표준출력에 쓰기
31         printf(stdout, "%d%18s%d%8d%8d\n",
32             score.number, score.name, score.mid, score.final, score.quiz);
33         fread(&score, sizeof(pscore), 1, f);
34     }
35     printf("%s\n", "-----");
36     fclose(f);
37
38     return 0;
39 }
40 void printhead()
41 {
42     printf("%s\n", "-----");
43     printf("%8s%15s%10s%8s%8s\n", "번호", "이름", "중간", "기말", "퀴즈");
44     printf("%s\n", "-----");
45 }
```

6-13행 구조체 struct personscore 정의

14행 구조체 struct personscore 자료형 pscore로 정의

17행 frame에 읽을 파일이름인 score.bin 저장

27행 이진모드로 파일 f에서 구조체 pscore 자료를 하나 읽어 변수 score에 저장

31행 표준 출력으로 구조체 pscore 자료인 score에 출력

33행 다시 이진모드로 파일 f에서 구조체 pscore 자료를 하나 읽어 변수 score에 저장

36행 fclose()로 스트림을 닫음

알게 된 내용

1. 파일의 필요성, 입출력 스트림, 파일 스트림 개념
2. 텍스트 파일 입출력
3. 이진 파일 입출력

학생 교과목 포트폴리오를 만들며...

교과목 포트폴리오를 만들기 위해 어떤 식으로 구성을 해야 할지에 대해 고민을 많이 하였습니다. 그 결과 학습 목표에 중점을 맞춰 내용을 정리해 어떤 내용을 배우고 이해해야 하는지에 대해 파악하려고 노력했고 단원별로 알게 된 내용을 간단히 정리하며 한 번 더 생각하는 시간을 갖도록 하였습니다. 추가적으로 시각적 부분을 고려해 ppt로 직접 만든 그림 자료와 code를 넣는 방식으로 제작하였습니다.

단지 수업을 듣는 것보다 저만의 방식으로 요약하여 포트폴리오를 만드는 과정을 통해 생각보다 이해하지 못하고 넘겼던 부분이 많았던 것을 깨닫는 시간이 되었고 다시 공부하는 시간을 가지며 C언어가 머릿속에 좀 더 정리된 것 같아 도움이 많이 되었습니다.

마지막으로 이번 포트폴리오를 제작하며 아쉬웠던 점은 너무 많은 분량이 담겨 내용이 방대해진 점이며, 앞으로 모르는 부분에 중점을 맞춰서 분량은 적지만 어려운 내용 위주로 꼼꼼하게 정리를 한 방식으로 다시 제작해보고 싶습니다.

Thanks!

C language