# 컴퓨터정보공학과

20160774
중간고사 포트폴리오
2학년 e반 김창우

2020/03/16 ~ 2020/05/03

# 목차

1장

## A. 문자와 문자열

**문자**란? 프로그램에서 다루는 자료는 수 또는 문자와 문자열이다. 문자는
영어의 알파벳이나 한글의 한 글자를 작음 따옴표로 둘러싸서 'A'와 같이 표기하며 작은 따옴표에 의해 표기된 문자를 문자 상수라고한다.

**문자열**? 문자의 모임인 일련의 문자를 문자열(string)이라 한다. 문자열은 일련의 문자 앞 뒤로 큰 따옴표로 둘러싸서 "java"라고 칭할 수가 있다. 큰 따옴표에 의해 표기된 문자열을 문자열 상수라하는데 "A"처럼 문자 하나도 큰 따옴표로 둘러싸면 문자열 상수라하는데 'ABC'처럼 작은 따옴표 로 둘러싸도 문자가 될 수 없다.

**문자와 문자열의 선언**

C 언어에서 char형 변수에 문자를 저장하기 위해 문자의 모임인 문자 배열을 사용한다. 문자 배열을 선언한 각각의 원소에 문자를 저장할 수 있다. 그러나 문자열의 맞;막을 임하는 NULL 문자 '\0'은 마지막에 저장되어야 하고 문자열이 저장될 배열의 크기는 반드시 저장될 문자 수보다 1이 커야 한다는 규칙이 있다.

```
char ch = 'B';

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

배열 csharp의 크기를 3으로 선언하고 배열 csharp에 문자열 '#'을 선언하고 저장하려면 맨 뒤에 csharp 값은 '\0'을 선언하여야 한다.

```
char abcd[] = {'A','B','C','D', '₩0'};
```

배열 선언 시 문자열을 저장하는 방법으로는 문자 하나하나 저장 시 마지막에 널(NULL) 값인 '\0'을 입력하여야 저장이 가능하고 문제없이 출력 된다.

**함수 printf()를 사용한 문자와 문자열 출력**

형식 제어문자 %c로 문자를 출력하며 배열 이름 문자 포인터를 사용하여 형식 제어문자 %s로 문자열을 출력할 수 있다. 함수 puts(csharp)와 같이 사용하면 한 줄에 문자열을 출력 가능하고 printf(c)와 같이 바로 배열 이름을 인자로 사용해도 출력이 가능하다.

```c
// file: chararray.c
#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";        //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;
}
```

7번째 줄) char형 변수 ch 선언하여 문자 'A'저장
8번째 줄) printf()에서 %c로 문자 변수 ch출력 %d사용하여 문자 코드값 출력
11번째 줄) 문자 하나 하나 초기화할 경우에는 마지막에 '\0' 삽입
12번째 줄) printf()에서 %s로 문자배열 변수 이름 java 삽입
15번째 줄) printf()에서 %s로 문자배열 변수 이름 c 삽입
18번째 줄) printf()에서 %s로 문자배열 변수 이름 csharp를 삽입
21번째 줄) 문자열이 저장된 문자배열에서 %c로 문자배열 원소를 저장하여 문자 하나를 출력

**'\0'문자에 의한 문자열 분리**

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

```c
// file: commandarg.c
#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)  //(*p != '\0')도 가능
        printf("%c", *p++);
        printf("\n");

        return 0;
}
```

6번째 줄) char형 배열 변수 c 선언하여 문자열 "C C++ JAVA" 저장
7번째 줄) printf()에서 %s 문자 배열 c 출력 문자열 "C C++ JAVA" 출력
8번째 줄) c[5]에 '\0' 저장하여 "C C++\0Java" 저장되어 "C C++", "Java"로 나뉨
12번째 줄) c[5]에 ' '을 저장하면 "C C++ JAVA"이 복원됨
13번째 줄) char 포인터 변수 p를 선언하여 c의 첫 주소 저장
14,15번째 줄) 포인터가 p 값이 아니면 while 반복 실행, p의 주소 값을 1씩 증가

**다양한 문자 입출력**

**버퍼리처리 함수 getchar()**

버퍼처리 함수 getchar()는 문자의 입력에 사용되고 putchar()는 문자의 출력에 사용된다.
문자 입력을 위한 함수 getchar()는 라인 버퍼링 방식을 사용하므로 문자 하나를 입력해도 반응을 보이지 않다가 엔터키를 누르면 입력이 실행된다.

## 함수 getche()

버퍼를 사용하지 않고 문자를 입력하는 함수이며 버퍼를 사용하지 않으므로 문자 하나를 입력하면 함수 getche()가 실행된다. 즉 버퍼를 사용하지 않고 문자 하나를 바로바로 입력할 수 있고 헤더파일 conio.h를 삽입하여 실행해야한다.

## 함수 getch()

문자 입력을 위한 함수 getch()는 입력한 문자가 화면에 보이지 않아서 입력된 문자를 출력함 수로 따로 출력하지 않으면 입력 문자가 화면에 보이지 않는다. getch()는 버퍼를 사용하지 않는 문자 입력 함수이며 getche()와 마찬가지로 conio.h를 삽입하여 사용해야 한다.

```
1  // file: getche.c
2  #include <stdio.h>
3  #include <conio.h>
4
5  int main(void)
6  {
7          char ch;
8
9          printf("문자를 계속 입력하고 Enter를 누르면 >>\n");
10         while ((ch = getchar()) != 'q')
11                 putchar(ch);
12
13         printf("\n문자를 누를 때마다 두 번 출력 >>\n");
14         while ((ch = _getche()) != 'q')
15                 putchar(ch);
16
17         printf("\n문자를 누르면 한 번 출력 >>\n");
18         while ((ch = _getch()) != 'q')
19                 _putch(ch);
20         printf("\n");
21
22         return 0;
23 }
```

7번째 줄) 입력한 문자를 저장할 공간으로 char 형 변수 ch 선언

10번째 줄) 함수 getchar()로 입력한 문자를 변수 ch에 저장하여 'q'가 아니면 putchar(ch)를 실행 변수 ch에 'q'반복 종료라 출력 불가

14번째 줄) getche() 입력하여 변수 ch 저장 'q'가 아니면 putchar(ch)를 실행 'q'반복 종료라 출력이 불가 gtche() 입력하여 바로 처리 입력

18번째 줄) getche() 입력하여 변수 ch 저장 'q'가 아니면 putchar(ch)를 실행 'q'반복 종료라 출력이 불가 gtche() 입력하여 바로 처리 입력, 문자도 보이지 않음

## gets()와puts()

함수 gets()는 한 행 문자열 입력에 유용한 함수이며 puts()는 한 행에 문자열을 출력해 주는 함수이다. gets(), puts(), gets_s()를 사용하려면 헤더 파일 stdio.h 삽입하여 사용해야 한다. gets()는 엔터 키를 누를 때까지 한 행을 버퍼에 저장한 후 입력 처리하므로 함수 gets()는 마지막에 입력된 '\n'가 '\0'로 교체되어 인자인 배열에 저장되어서 한 행을 하나의 문자열로 간주하여야 한다.

```
char * gets(char * buffer);
```

함수 gets()는 문자열을 입력 받아 buffer에 저장하고 입력받은 첫 문자의 주소값을 반환한다.

```
char * gets_s(char * buffer, size_t sizebuffer);
```

sizebuffer는 정수형으로 buffer의 크기를 입력한다.

```
int puts(const char * str);
```

인자인 문자열 str에서 마지막 '\0' 문자를 개행 문자인 '\n'로 대체하여 출력한다.

```c
// file: gets.c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

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

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

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

        return 0;
}
```

7번째 줄) 한 줄에 입력되는 모든 문자열이 입력되도록 충분한 크기의 문자배열 line[101] 선언

10번째 줄) 함수 gets(line) 호출로 한 줄 전체 입력 받음

11번째 줄) 함수 puts(line) 호출로 한 줄 전체를 출력

14번째 줄) 함수 gets_s(line, 101) 호출로 한 줄 전체를 입력 받음

15번째 줄) 함수 puts(line) 호출로 한 줄 전체를 출력



## B. 문자열 관련 함수

다양한 문자열 라이브러리 함수에는 문자열 비교와 복사 문자열 연결 등과 같은 다양한 문자열 처리는 헤더파일 string.h에 함수 원형으로 선언된 라이브러리 함수로 제공된다. 함수에서 사용되는 자료형 size_t는 비부호 정수형이며 void *는 아직 정해지지 않은 다양한 포인터를 의미한다.

문자의 배열 관련 함수는 헤더파일 string.h에 함수원형이 정의되어 있다.

| 함수원형 | 설명 |
|---|---|
| void *memchr (const void *str, int c, size_t n) | 메모리 str에서 n바이트까지 문자 c를 찾아 그 위치를 반환 |
| int *memcmp(const void *str1, const void *str2,size_t n) | 메모리 str1과 str2를 첫 n 바이트를 비교하여 같으면 0, 다르면 음수 또는 양수로 반환 |
| void*memcmp(void *dest, const void *src, size_t n) | 포인터 src 위치에서 dest에 n 바이트를 복사한 후 dest 위치 반환 |
| void*memmove(void *dest, const void *src, size_t n) | 포인터 src 위치에서 dest에 n 바이트를 복사한 후 dest 위치 반환 |
| void *meset(void*str,m int c, szie_t n) | 포인터 src 위치에서 n 바이트까지 문자 c를지정한 후 src 위치 반환 |
| size_t strlen(const char *str) | 포인터 src 위치에서부터 널 문자를 제외한 문자열의 길이 반환 |

## 함수 strcmp()

문자열 비교와 복사 문자열 연결 등과 같은 다양한 문자열 처리는 헤더파일 string.h에 함수 원형으로 선언된 라이브러리 함수로 제공되며 대표적인 문자열 처리 함수인 strcmp()와 strncmp()는 문자열을 비교하는 함수로 쓰인다.

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

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

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

문자열을 같은 위치의 문자를 앞에서부터 다를 때까지 비교하여 최대 n까지만 한 뒤 같으면 0을 반환하고 앞이 크면 양수를 뒤가 크면 음수를 반환한다.

## 함수 strcpy()

함수 strcpy()와 strncpy()는 문자열을 복사하는 함수이며 함수 strcp()는 앞 인자 문자열 dest에 뒤 인자 문자열 source를 복사한다. 첫 번째 인자인 dest는 복사 결과가 저장될 수 있도록 충분한 공간을 확보해야 하고 복사되는 최대 문자 수를 마지막 인자 maxn으로 지정

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

앞 문자열 dest에 뒤 문자열 null 문자를 포함한 source를 복사하여 문자열을 반환하고 앞 문자열은 수정되지만 뒤 문자열은 수정될 수 없다.

```
char * strcpy(char * dest, const char * source, size-t maxn);
```

앞 문자열 dest에 뒤 문자열 source에서 n개 문자를 복사하여 문자열을 반호나하고 지정된 maxn이 source의 길이보다 길면 나머지는 모두 널 문자가 복사된다. 문자열은 수정되지만 뒤 문자열은 수정될 수 없다.

```
errno_t strcpy_s(char * dest, size_t sizedest, const char * source);
errno_t strcpy_s(char * dest, size_t sizedest, const char * source, size_t maxn);
```

두 번째 인자인 sizedest는 정수형으로 dest의 크기를 입력하고 반환형 errno_t는 정수형이며 반환값 은 오류번호로 성공하면 0을 반환하고 strcpy_s()와 strncpy_s() 함수들이 자주 사용된다.

## 함수 strcat()

함수 strcat()는 하나의 문자열 뒤에 다른 하나의 문자열을 연이어 추가해 사용하게 해줄 수 있고 함수 strcat()는 앞 문자열에 뒤 문자열의 null 문자까지 연결하여 앞의 문자열 주소를 반환하게 해주는 함수이다

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

앞 문자열 dest에 뒤 문자열 source를 연결하여 저장한 뒤 이 연결된 문자열을 반환하고 뒤 문자열은 수정될 수 없다.

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

앞 문자열 dest에 뒤 문자열 source중에서 n개의 크기만큼을 연결해 저장한 뒤 이 연결된 문자열을 반환하고 뒤 문자열은 수정될 수 없고 지정한 maxn이 문자열 길이보다 크면 null 문자까지 연결해준다.

```
errno_t strcat_s(char * dest, size_t sizedest, const char * source);
errno_t strcat_s(char * dest, size_t sizedest, const char * source, size_t maxn );
```

두 번째 인자인 sizedest는 정수형으로 dest의 크기를 입력하고 반환형 errnon_t는 정수형이며 반환값은 오류번호로 성송하면 0을 반환하고 strcat_s()와 strncat_s() 함수들이 자주 사용된다.

## 함수 strtok()

문자열에서 구분자인 문자를 여러 개 지정하여 토큰을 추출하는 함수이며 strtok()는 첫 번째 인자인 str은 토큰을 추출할 대상인 문자열이며 두 번째 인자인 delim은 구분자로 문자의 모임인 문자열이다. 첫 번째 인자인 str은 문자배열에 저장된 문자열을 사용하여야하고 str은 문자열 상수를 사용할 수 없다.

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

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

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

마지막 인자인 context는 함수 호출에 사용되는 위치 정보를 위한 인자이며 Visual c++에서는  strtok_s()의 사용을 권장한다.

1장

## C. 여러 문자열 처리

### Ⅰ.문자 포인터 배열

여러 개의 문자열을 처리하는 하나의 방법을 문자 포인터 배열이라 한다. 하나의 문자 포인터가 하나의 문자열을 참조할 수 있으므로 문자 포인터 배열은 여러 개의 문자열을 참조할 수 있다.

```
char *pa[] = {"java", "c#", "c++"};
printf("%s ", pa[0]); printf("%s ", pa[1]); printf("%s₩n", pa[2]);
```

문자 포인터 배열을 사용해서 처리하는 방법인데 먼저 배열의 크기 문자열 개수인 3을 지정한 뒤 문자 포인터 배열 pa를 사용하고 각 문자열을 출력하려면 pa[i]로 형식제어문자 %s를 이용하여 구현한다.

### Ⅱ.이차원 문자 배열

문자 포인터 배열 방법 말고 여러 개의 문자열을 처리하는 다른 방법은 이차원 문자 배열이다 배열선언에서 이차원 배열의 열 크기는 문자열 중에서 가장 긴 문자열의 길이보다 1크게 지정한뒤 사용하여야 한다.

```
char ca[][5] = {"java", "c#", "c++"};
printf("%s ", ca[0]); printf("%s ", ca[1]); printf("%s₩n", ca[2]);
```

이차원 문자 배열을 사용하여 처리하는 방법이다. 가장 긴 문자열 “java”보다 1이 큰 5를 2차원 배열의 열 크기로 지정하고 이차원 배열 행의 크기는 문자열 수에 해당하므로 3으로 지정하거나 공백으로 둔다. 문자 이차원 배열 ca를 이용하여 출력 하려 할때 ca[i] 형식제어문자 %s를 이용하여 구현한다.

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5          char* pa[] = { "JAVA", "C#", "C++" };
6          char ca[][5] = { "JAVA", "C#", "C++" };
7
8          //각각의 3개 문자열 출력
9          //pa[0][2] = 'v';       //실행 오류 발생
10         //ca[0][2] = 'v';       //수정 가능
11         printf("%s ", pa[0]); printf("%s ", pa[1]); printf("%s\n", pa[2]);
12         printf("%s ", ca[0]); printf("%s ", ca[1]); printf("%s\n", ca[2]);
13
14         //문자 출력
15         printf("%c %c %c\n", pa[0][1], pa[1][1], pa[2][1]);
16         printf("%c %c %c\n", ca[0][1], ca[1][1], ca[2][1]);
17
18         return 0;
19 }
```

5번째 줄) 문자 포인터 배열 pa를 선언하여 초기값으로 세 개의 문자열을 저장
6번째 줄) 열 크기가 3인 이차원 문자배열 ca를 선언하여 초기값으로 “JAVA”, “C#”, “C++”을 저장

11번째 줄) pa[0],pa[1],pa[2] 문자열을 가리키는 포인터 함수 printf()에서 %s로 문자열 출력 가능

12번째 줄) ca[0],ca[1],ca[2] 문자열을 가리키는 포인터 함수 printf()에서 %s로 문자열 출력 가능

15,16번째줄) pa[0],pa[1],pa[2] ca[0],ca[1],ca[2] 모두 세 문자열에서 두 번째 문자가 저장된 변수로 printf()에서 %c로 문자 출력 가능

### 명령행 인자

명령행 인자는 입력하는 문자열을 프로그램으로 전달하는 방법이 명령행 인자라하는데 프로그램에서 명령행 인자는 main() 함수의 인자로 기술된다.
프로그램에서 명령랭 인자를 받으려면 main() 함수에서 두 개의 인자 argc와 argv를(int argc, char *argv[])로 기술하고 매개변수 argc는 명령행에서 입력한 문자열의 수이며 argv[]는 명령행에서 입력한 문자열을 전달 받는 문자 포인터 배열이라한다.

```
1  #include <stdio.h>
2
3  int main(int argc, char* argv[])
4  {
5          int i = 0;
6
7          printf("%ÇÇà ¸í·ÉÇà ÀÎÀÚ(command line arguments) >>\n");
8          printf("argc = %d\n", argc);
9          for (i = 0; i < argc; i++)
0                  printf("argv[%d] = %s\n", i, argv[i]);
1
2          return 0;
3  }
```

3번째 줄) 명령행 인자를 처리하려면 main()의 매개변수를 “int argc, char *argv[]”로 기술 argc에 인자의 수는 argv인자인 여러 개의 문자열 포인터가 저장된 배열에 전달

8번째 줄) 인자의 수인 argc를 출력

9번째 줄) 제어 변수인 i는 0부터 argc보다 작을 때까지 반복

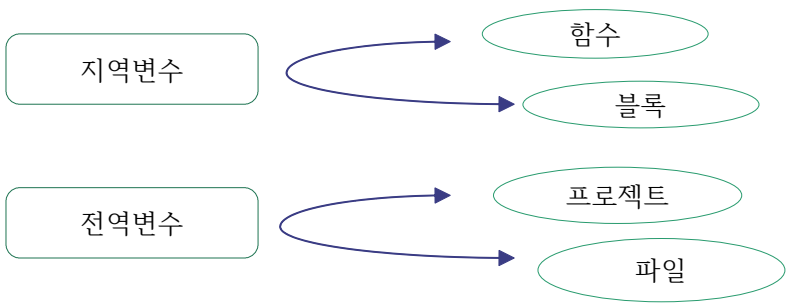
10번째 줄) 제어 변수인 i를 사용하고 문자열 배열 argv[i]를 참조하여 명령행 인자인 각각의 문자열 출력

## 2장 변수 유효범위

### A. 전역변수와 지역변수

### Ⅰ.변수 범위??

변수의 참조가 유효한 범위를 변수의 유효 범위(scope)라 하는데 변수의 유효 범위는 크게 지역 유효범위 저녁 유효 범위로 나눌 수 있다. 지역 유효 범위는 함수 또는 블록 내부에서 선언되어 그 지역에서 변수의 참조가 가능한 범위이고 전역 유효 범위는 소위 2가지로 구분되는데 1. 하나의 파일에서만 변수의 참조가 가능한 범위이고 2. 프로젝트를 구성하는 모든 파일에서 변수의 참조가 가능한 범위이다.



## Ⅱ. 지역변수

지역변수는 함수, 블록에서 선언된 변수이며 내부 변수 또는 자동변수라 부른다. 지역변수는 선언 문장 이후에 함수나 블록의 내부에서만 사용이 가능하며 다른 함수나 블록에서는 사용될 수 없다 매겨 변수도 함수 전체에서 사용 가능한 지역변수와 같으며 선언 후 초기화하지 않으면 쓰레기 값이 저장되므로 주의해야 하는 변수이다.

지역변수가 할당되는 메모리 영역을 스택이라 하는데 변수가 선언된 부분에서 자동으로 생성되고 함수나 블록이 종료되는 순간 메모리에 자동 제거가 되어서 이런 현상을 자동변수라 칭한다.

```c
#include <stdio.h>

void sub(int param);

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

	//m, sum은 for 문 내부의 블록 지역변수
	for (int m = 0, sum = 0; m < 3; m++)
	{
		sum += m;
		printf("\t%d %d\n", m, sum);
	}

	printf("%d\n", n); //n 참조 가능
	//printf("%d %d\n", m, sum); //m, sum 참조 불가능

	//함수호출
	sub(20);

	return 0;
}

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

8번째 줄) 지역 변수 n 선언하여 10 저장한 뒤 변수 n의 유효 범위를 8번째부터 26번째 줄까지 저장

12번째 줄) for 문 내부 블록 지역 함수 m과 sum을 선언 변수 m, sum은 유효 범위를 12번째 줄부터 16번째까지

28번째 줄) 매개변수 param은 지역변수로 유형 범위를 29번째 줄부터 34번째 줄까지

31번째 줄) 지역 변수 local의 유효 범위를 31번째 34번째까지

## Ⅲ. 전역변수

전역변수는 함수 외부에서 선언되는 변수이다. 전역변수는 외부 변수라고도 부르는데 전역변수는 일반적으로 프로젝트의 모든 함수나 블록에서 참조할 수 있다.

전역변수는 선언되면 자동으로 초깃값이 자료형에 맞는 0으로 지정되는데 정수형은 0, 문자형은 null 문자인 '\0' 실수형은 0.0 포인터 형은 NULL 값으로 저장이 된다. 함수나 블록에서 전역변수와 같은 이름으로 지역변수를 선언할 수 있는데 함수 내부, 블록에서 이름을 참조하면 지역변수로 인식하는데 지역변수와 동일한 이름의 전역변수는 참조할 수 없다. 또 전역변수는 프로젝트의 다른 파일에서도 참조가 가능한데 그러나 다른 파일에서 선언된 전역변수를 참조하려면 키워드 extern을 사용하여 다른 파일에서 선언된 전역변수임을 선언하고 사용 가능하다.

### Ⅲ-Ⅰ. 전역변수 extern

extern을 사용한 참조 선언 구문은 변수 선언 문장 맨 앞에 extern을 넣는 구조이며 extern을 참조 선언 구문에서 자료형은 생략할 수 있으며 키워드 extern을 사용한 변수 선언은 새로운 변수를 선언하는 것은 아니며 이미 존재하는 전역변수의 유효 범위를 확장하는 것이다.

```c
#include <stdio.h>

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

//전역변수 선언
double PI = 3.14;
int gi;

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

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

	return 0;
}

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

7번째 줄) 전역 변수  PI 선언하여 3.14를 저장
변수 PI 유효 범위를 전체로 저장

8번째 줄) 전역 변수 gi 선언하여 기본값인 0을저장하고 변수 gi 유효 범위를 전체로 저장

13번째 줄) 지역 변수 r 선언

15번째 줄) 지역 변수 PI 선언 전역 변수와 충돌하여 지역 변수가 우선일 저역 변수 참조 불가능

20.21번째 줄) 지역 변수 PI 출력,
전역 변수인 gi의 기본값 출력

## B. 정적 변수와 레지스터 변수

### Ⅰ.기억부류??

변수 선언의 위치에 따라 변수는 전역과 지역으로 나뉘는데 그중에서도 4가지의 기억 부류인 auto, register, static, exetern에 따라 할당되는 메모리 영역이 결정되고 메모리의 할당과 제거 시기가 결정된다. 모든 기억 부류는 전역 변수 또는 지역 변수로 이루어져 있고 전역과 지역은 변수의 선언 위치에 따라 결정되나 기억 부류는 키워드 auto, register, static, extern에 의해 구분되므로 구분하기는 쉬우나 자동변수인 auto는 일반 지역변수로 생략될 수 있으므로 주의를 필요로 한다.

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

기억 부류 auto와 register는 지역변수에만 이용이 가능하고 static은 지역과 전역 모든 변수에 이용이 가능하다 extern은 전역변수에만 사용 가능하다. 기억 부류 사용 구문은 변수 선언 문장에서 자료형 앞에 하나의 키워드를 넣는 방식인데 키워드 extern를 제외하고 나머지 3개의 기억 부류의 변수 선언에서 초깃값을 저장할 수 있다. 키워드 auto는 지역변수 선언에서 사용되며 생략이 가능하다. auto는 함수에 선언된 모든 변수가 생략된 자동변수이다.

## Ⅱ. 키워드 register

register는 일반적으로 변수는 메모리에 할당되는데 레지스터 변수는 변수의 저장공간이 일반 메모리가 아니라 CPU내부의 레지스터에 할당되는 변수이다.
레지스터 변수는 키워드 register를 자료형 앞에 넣어 선언하고 지역변수에만 이용이 가능하며 레지스터는 CPU내부에 있는 기억장소이므로 일반 메모리보다 빠르게 참조될 수 있어서 레지스터 변수는 처리 속도가 빠른 장점이 있다. 그러나 일반 메모리에 할당되는 변수가 아니므로 주소연산자 &를 사용할 수 없어서 레지스터 변수에 주소연산자 &을 사용하면 오류가 발생

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

int main(void)
{
	//레지스터 지역변수 선언
	register int sum = 0;

	//메모리에 저장되는 일반 지역변수 선언
	int max;
	printf("양의 정수 입력 >> ");
	scanf("%d", &max);

	//레지스터 블록 지역변수 선언
	for (register int count = 1; count <= max; count++)
		sum += count;

	printf("합: %d\n", sum);

	return 0;
}
```

7번째 줄) 레지스터 변수 sum 선언하면서 초기값으로 0 저장, 초기값이 없으면 쓰레기값 저장

10번째 줄) 일반 지역변수 max 저장 초기값이 없으면 쓰레기값 저장

12번째 줄) max에 표준입력으로 정수 저장

15번째 줄) 레지스터 블록 지역변수 count 선언하면서 초기값으로 1 저장 count는 max까지 반복하면서 sum+= count를 실행

16번째 줄) sum+= count에서 count는 1에서 max까지 변하므로 1에서 max까지의 합을 저장

## Ⅲ.정적 변수

### Ⅲ-Ⅰ 키워드 static

변수 선언에서 자료형 앞에 키워드 static을 넣어 정적 변수(static variable)를 선언할 수 있는데 정적 변수는 초기 생성된 이후 메모리에서 제거되지 않으므로 지속적으로 저장 값을 유지하거나 수정할 수 있는 특성이 있고 프로그램이 시작되면 메모리에 할당되고 프로그램이 종료되면 메모리에서 제거된다. 정적 변수는 초깃값을 지정하지 않으면 자동으로 자료형에 따라 0이나 '\0' 또는 NULL 값이 저장된다. 그러나 초기화는 단 한 번만 수행되며 상수로만 가능하다.
EX) static int svar = 1;
전역변수 선언 시 키워드 static을 가장 앞에 붙이면 정적 전역변수가 됨
지역변수 선언 시 키워드 static을 가장 앞에 붙이면 정적 지역변수가 됨

정적변수는 정적 지역변수와 정적 전역변수로 나눌 수 있다.

## Ⅲ.정적 변수

### Ⅲ-Ⅱ 정적 지역변수

함수나 블록에서 정적으로 선언되는 변수는 정적 지역변수인데 정적 지역변수의 유효 범위는 선언된 블록 내부에서만 참조 가능하며 정적 지역변수는 함수나 블록을 종료해도 메모리에서 제거되지 않고 계속 메모리에 유지 관리되는 특징과 변수 유효 범위는 지역변수와 같으나 할당된 저장공간은 프로그램이 종료되어야 메모리에서 제거되는 전역변수 특징도 갖고 있다.

## Ⅲ.정적 변수

### Ⅲ-Ⅲ 정적 전역변수

함수 외부에서 정적으로 선언되는 변수가 정적 전역변수라하는데 일반 전역변수는 파일 소스가 다르더라도 extern을 사용하여 참조가 가능한데 정적 전역변수는 선언된 파일 내부에서만 참조가 가능한 변수이다. 정적 전역변수는 extern에 의해 다른 파일에서 참조가 불가능하다. 정적 전역변수의 사용은 모든 함수에서 공유할 수 있는 저장공간을 이용할 수 있는 장점이 있으나 한곳이 잘못되면 모든 함수에 영향을 미치는 단점이 있다. 프로그램이 크고 복잡하면 전역변수의 사용은 원하지 않는 전역변수의 수정과 같은 부작용의 위험성이 항상 존재한다.

```
1  void teststatic()
2  {
3          //정적 전역변수는 선언 및 사용 불가능
4          //extern svar;
5          //svar = 5;
6  }
7
8  void testglobal()
9  {
10         //전역변수는 선언 및 사용 가능
11         extern gvar;
12         gvar = 10;
13 }
```

1번째 줄) 함수 testatic() 구현 헤드를 선언

4번째 줄) 정적 전역변수 svar는 다른 파일에서 사용 불가하여 실행 시 오류 발생

5번째 줄) 정적 전역변수 svar는 사용 불가

8번째 줄) 함수 testglobal() 구현 헤드 선언
11번째 줄) 다른 파일에 정의된 전역변수 gvar는 이 파일에서 extern을 사용해 선언 한 후 사용 가능

12번째 줄) 전역변수 gvar 사용 가능하므로 10으로 저장함

## C. 메모리 영역과 변수 이용

### Ⅰ.메모리영역??

메인 메모리의 영역은 프로그램 실행 과정에서 데이터 영역, 힙 영역, 스택 영역 세 부분으로 나뉘는데 메모리 영역은 변수의 유효범위(scope)와 생존기간에 결정적 역할을하며 변수는 기억부류에 따라 할당되는 메모리 공간이달라진다.
기억부류는 변수의 유효범위와 생존기간을결정하는데 기억부류는 변수의 저장공간의 위치가 **데이터 영역 힙 영역 스택 영역**인지 결정하며 초기값도 결정해준다. **데이터 영역**은 전역변수와 정적변수가 할당되는 저장공간과 메모리 주소가 낮은 값에서 높은 값으로 저장 장소가 할당된다. **힙 영역**은 데이터 영역과 스택 영역 사이에 위치하며 동적 할당되는 변수가 할당되는 저장공간이다. **스택 영역**은 함수 호출에 의한 형식 매개변수 함수 내부의 지역변수가 할당되는 저장공간이고 힙 영역과 더불어 프로그램이 실행되면서 영역 크기가 계속적으로 변한다.

### Ⅱ. 변수의 이용

일반적으로 전역변수의 사용을 자제하고 지역변수를 주로 이용하는데 4가지의 특성에 맞는 변수를 이용한다.

1. 실행 속도를 개선하고자 하는 경우에 제한적으로 특수한 지역변수인 레지스터 변수를 이용

2. 함수나 블록 내부에서 함수나 블록이 종료되더라도 계속적으로 값을 저장하고 싶을 때는 정적 지역변수를 이용한다.

3. 해당 파일 내부에서만 변수를 공유하고자 하는 경우는 정적 전역변수를 이용한다.

4. 프로그램의 모든 영역에서 값을 공유하고자 하는 경우는 전역변수를 이용한다. 가능하면 전역 변수의 사용을 줄이는 것이 프로그램의 이해를 높일 수있으며 발생할 수 있는 프로그램 문제를 줄일 수 있다.

**변수가 정의되는 위치에 따라 전역변수와 지역변수**는 데이터 영역에 할당되는 전역변수와 정적 변수는 프로그램 시작 시 메모리가 할당되고 프로그램 종료시 메모리에서 제거되고
스택 영역과 레지스터에 할당되는 자동 지역변수와 레지스터 변수는 함수 또는 블록 시작 시 메모리가 할당되고 함수 또는 블록 종료 시 메모리에서 제거된다.

**변수의 종류에 따른 생존기간과 유효 범위**는 전역변수와 정적 변수는 모두 생존 기가이 프로그램 시작 시에 생성되어 프로그램 종료 시에 제거되고 자동 지역변수와 레지스터 변수는 함수가 시작되는 시점에서 생성되어 함수가 종료되는 시점에서 제거된다.

```c
#include <stdio.h>

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

/* 전역변수*/
int global = 10;
/* 정적 전역변수*/
static int sglobal = 20;

int main(void)
{
        auto int x = 100;   /* main() 함수의 자동 지역변수*/

        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)
{
        /* infunction() 함수의 자동 지역변수*/
        auto int fa = 1;
        /* infunction() 함수의 정적 지역변수*/
        static int fs;

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

7번째 줄) 전역변수 global 선언하면서 10으로 초기화 함수 외부이므로 전역변수 이 위치에서 사용하려면 extern int global; 선언

9번째 줄) 정전 전역변수  sglobal 선언 20으로 초기화 함수 외부여서 전역변수이나 파일 위치 이하에서 사용가능

13번째 줄) 변수 x는 자동 지역변수로 초기화 100 저장 함수 main()에서만 사용 가능

15번째 줄) 변수 전역변수 global,정적 전역변수 sglobal, 지역변수 x를 출력

19번째 줄) 변수 전역변수 global,정적 전역변수 sglobal, 지역변수 x를 출력

27번째 줄) 변수 fa는 자동 지역변수로 초기화로 1 저장 함수 infunction()에서 사용가능

29번째 줄) 변수 fs는 정적 지역변수로 첫 번째 호출에서 초기화가 없어서 기본값 0저장 값은 유지됨 함수 infunction()에서만 사용 가능

# 3장 변수 유효범위

## A. 구조체와 공용체

### Ⅰ.구조체??

구조체는 정수 문자 실수 포인터들의 배열들을 묶어 하나의 자료형으로 이용하는 것이다. 구조체는 프로그램에서 서로 다른 자료 항목이 긴밀하게 연계해 주고 하나의 단위로 관리해 주는 역할을 하는데 즉 연관성이 있는 서로 다른 개별적인 자료형의 변수들을 하나의 단위로 묶은 새로운 자료형이라 한다. 구조체는 연관된 멤버로 구성되는 통합 자료형으로 대표적인 유도 자료형이며 기존 자료형으로 새로이 만들어진 뜻도 품고 있다.

## Ⅱ.구조체 정의??

자료형 int 나 double과 같은 일반 자료형은 자료형을 사용할 수 있으나 구조체를 자료형으로 사용하려면 구조체로 정의하고 구조체를 만들 구조체 틀 또한 정의 내려 사용해야 한다.
구조체의 정의는 변수의 선언과는 다른 것으로 변수 선언에서 이용될 새로운 구조체 자료형을 정의하는 구문이며 멤버 선언 구문은 하나의 문장이므로 반드시 세미콜론으로 종료하고 각 구조체 멤버의 초깃값을 대입할 수 없다. 멤버가 ex) int credit; int hour;처럼 같은 자료형이 연속적으로 놓일 경우 콤마 연산자를 사용해 int credit, hour;로도 가능하다.

```
struct lecture
{
   char name[20];
   int credit;
   int hour;
}
```

구조체 태그 이름을 lecture라 선언한 뒤 자료형에 대한 변수명을 각 각 선언한뒤 마지막 멤버 구문들에는 ;을 필히 선언

## Ⅱ-Ⅰ.구조체 변수 선언

구조체가 정의되었다면 변수 선언이 가능한데 새로운 자료형 struct account 형 변수 mine을 선언하려면 struct account mine; 으로 선언한다. 구조체를 선언 하는 다른 방법은 문장으로 구조체 struct account를 정의하면서 동시에 변수 myaccount는 구조체 struct account형의 변수로 선언이되고 문장 이후 struct account도 새로운 자료형으로 사용 된다.

```
struct lecture
{
   char name[12];
   int actnum;
   double balance;
} myaccount;

struct account youraccount;
```

구조체 태그이름 변수명을 선언하고 변수 myaccount는 struct account형 변수로 선언하고 마찬가지로 변수 youraccount 도 struct account형 변수로 선언한다.

## Ⅱ-Ⅱ.구조체 변수 선언

구조체 멤버로 이미 정의된 다른 구조체 형 변수와 자기 자신을 포함한 구조체 포인터 변수 사용이 가능하다.

```c
#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, 300000 };

	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);
}
```

5번째 줄) 구조체 struct date를 정의하기 위한 문장을 5행부터 10행까지 선언

7번째 줄) 구조체 date 멤버인 년 year을 위한 선언 문으로 마지막에 ;을 반드시 선언

8번째 줄) 구조체 date 멤버인 월 month을 위한 선언 문으로 마지막에 ;을 반드시 선언

9번째 줄) 구조체 date 멤버인 일 day을 위한 선언 문으로 마지막에 ;을 반드시 선언

15번째 줄) 구조체 account 내부 멤버 구조체 struct date 형으로 변수 open을 선언함

23번째 줄) 구조체 struct account 형인 me를 선언하여 초기화 한 뒤 모든 멤버를 모두 초기화

## A. 구조체와 공용체

### Ⅲ.공용체??

공용체는 동일한 저장 장소에 여러 자료형을 저장하는 방법으로 공용체를 구성하는 멤버를 한 번에 한 종류만 저장하고 참조할수 있다. 공용체(union)는 서로 다른 자료형의 값을 동일한 저장공간에 저장하는 자료형인데 선언방법으로는 union을 struct로 사용하며. 그 외에는 구조체 선언 방법과 동일하다. 공용체 변수의 크기는 멤버 중 가장 큰 자료형의 크기로 정해지고 모든 멤버가 동일한 저장 공간을 사용하므로 동시에 여러 멤버의 값을 동시에 저장하여 이용할 수 없으며 마지막에 저장된 단 하나의 멤버 자료값만을 저장한다. 공용체의 초기화 값은 공용체 정의 시 처음 선언한 멤버의 초기값으로만 저장이 가능하다.

```c
typedef union data uniondata;

uniondata data2 = {'A'};

uniondata data3 = data2;
```

첫 멤버인 char형으로만 초기화가 가능하여 컴파일 시 경고가 발생할 수도 있고 다른 변수로 초기화도 가능하다. 또한 초깃값으로 동일한 유형의 다른 변수의 대입도 가능하다

## Ⅲ-Ⅰ.공용체 멤버 접근??

```
uniondata  data2;

data2.ch = 'A';
data2.cnt = 100;
data2.real= 3.80;
```

공용체 변수로 멤버를 접근하기 위해서는 구조체와 같이 접근 연산자 .를 사용한다. 유형이 char인 ch를 접근하면 8바이트 중에서 첫 1바이트만 참조하고 int인 cnt를 접근하면 전체 공간의 첫 4바이트만 참조한다. 또한 double인 real을 접근하면 8바이트 공간을 모두 참조할 수 있다. 공용체 멤버는 참조가 가능하나 항상 마지막에 저장한 멤버로 접근해야 원하는 값을 얻을 수 있다.

```c
#include <stdio.h>

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

int main(void)
{
        union data data2 = { 'A' };     //첫 멤버인 char형으로만 초기화 가능
        //union data data2 = {10.3};//컴파일 시 경고 발생
        union data data3 = data2;       //다른 변수로 초기화 가능

        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;
}
```

4번째 줄) char, int, double 자료형 하나를 동시에 저장할 수 있는 8바이트 공간의 공용체 data를 정의. 4행에서 9행까지 선언.동시에 변수 datal을 선언, 변수 datal은 모든 파일에서 사용 가능

13번째 줄) 공용체 union data 형으로 변수 data2를 선언하여 초기값으로 문자 'A'를 저장하고 공용체의 첫 멤버인 char형으로만 초기화 가능

14번째 줄) 공용체의 세 번째 멤버인 double 형으로는 초기화가 불가능하다.

15번째 줄) 공용체 union data 형으로 변수 data3를 선언하면서 초기값으로 같은 유형의 변수를 대입

17번째 줄) 공용체 union data 자료형과 변수의 크기인 8을 출력

20번째 줄) 공용체인 data1에서 자료형 char인 멤버 ch에 문자 'a'를 저장,

21번째 줄) 공용체인 data1에서 모든 멤버를 출력한뒤 data1.ch 만 정확히 출력

23번째 줄) 공용체인 data1에서 자료형 int인 멤버 cnt에 정수 100을 저장

26번째 줄) 공용체인 data1에서 자료형 double인 멤버 real에 실수 3.156759를 저장함

27번째 줄) 공용체인 data1에서 모든 멤버를 출력해 보나 data1.real만 정확히 출력한다.

## B. 자료형 재정의

### Ⅰ.자료형 재정의 typedef

typedef는 이미 사용되는 자료 유형을 다른 새로운 자료형 이름으로 재정의할 수 있도록 하는 키워드인데 문장 typedef int profit;는 int와 같은 자료형으로 새롭게 정의하는 문장이다. 이처럼 프로그램의 시스템 간 호환성과 편의성을 위해 자료형을 재정의를 한다. 문장 typedef는 일반 변수와 같이 사용 범위를 제한하는데 함수 내부에서 재정의되면 선언된 이후의 함수에서만 이용이 가능하다. 만약 함수 외부에서 재정의된다면 재정의된 이후 재정의된 파일에서 이용이 가능하다.

```c
#include <stdio.h>

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

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

        //함수 내부에서 정의된 자료형은 이후 함수내부에서만 사용 가능
        typedef int profit;
        //새로운 자료형 profit 사용
        profit month = 4600000;

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

        return 0;
}

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

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

4번째 줄) buget은 int와 같은 자료형이고 위치가 함수 외부이므로 자료형 buget은 이 위치 이후 파일에서 사용 가능

9번째 줄) buget은 int와 같은 자료형으로 변수 year를 buget으로 선언하여 초기값을 대입

14번째 줄) profit는 int와 같은 자료형으로 변수 month를 profit으로 선언하여 초기값 대입

24번째 줄) 자료형 buget은 함수 test()에서 사용 가능

27번째 줄) 자료형 profit은 함수 test()에서 사용 불가

### Ⅱ.구조체 자료형 재정의

구조체 자료형을 사용할 때 struct date는 항상 키워드 struct를 사용하는데 typedef를 사용하여 구조체를 한 단어의 새로운 자료형으로 정의하면 이러한 과정들을 생략할 수 있고 구조체 struct date가 정의된 상태에서 typedef를 사용하여 구조체 struct data를 date를 재정의가 가능하다.

```c
typedef struct
{
  char title[30];
  char company[30];
  char kinds[30];
  date release;

} software
```

typedef을 새로운 자료형으로 software 형을 정의한뒤 이후에는 software를 구조체 자료형으로 변수 선언에 사용이 가능하다. 구조체 태그이름도 생략은 가능한데 구조체 software형은 멤버로 구조체 date형 release를 갖을 수 있다.

## C. 구조체와 공용체의 포인터와 배열

### Ⅰ.구조체 포인터

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

```
lecture os = {"운영체제", 2, 3, 3};
lecture *p = &os;
```

변수 os를 선언후 lecture 포인터 변수 p에 &os 를 저장하면 포인터 p로 구조체 변수 os 멤버 참조가 가능하게 만들어진다.

### Ⅰ-Ⅱ.공용체 포인터

공용체 변수도 포인터 변수 사용이 가능하다 공용체 포인터 변수로 멤버를 접근하려면 접근연산자 ->을 사용하고

```
union data
{
  char ch;
  int  cnt;
  double real;
} value, *p ;

p = &value;
p ->ch = 'a';
```

포인터 p에 value의 주소값을 저장하고 value를 가리키는 포인터 p를 선언하여 p가 가리키는 공용체 멤버 ch에 'a'를 저장하면 value.ch = 'a';와 동일한 문장이된다.

### Ⅰ-Ⅲ.구조체 배열

다른 배열과 같이 동일하 구조체 변수가 여러 개 필요하면 구조체 배열을 선언하여 이용할수 있는데 구조체 배열의 초기값 지정 구문에서는 중괄호가 중첩된다. 외부 중괄호는 배열 초기화의 중괄호이며 내부 중괄호는 배열원소인 구조체 초기화를 위한 중괄호로 사용된다

```
#include <stdio.h>

struct lecture
{
        char name[20];  //강좌명
        int type;               //강좌구분
        int credit;             //학점
        int hours;              //시수
};
typedef struct lecture lecture;

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

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

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

        printf("배열크기: %d\n\n", arysize);
        printf("%12s   %12s %6s %6s\n", head[0], head[1], head[2], head[3]);
        printf("-------------------------------------------------\n");

        for (int i = 0; i < arysize; i++)
                printf("%16s %10s %5d %5d\n", course[i].name,
                        lectype[course[i].type], course[i].credit, course[i].hours);

        return 0;
```

18번째 줄) 구조체 struct의 배열을 선언하면서 바로 초기값을 대입하고 배열 크기는 지정하지 않고 초기값을 5로 지정

24번째 줄) 배열 크기를 계산하여 변수 arysize에 저장

26 27번째 줄) 배열 크기와 제목을 출력

30번째 줄) 첨자 I는 0에서 arysize-1까지 실행하게 구현

31 32번째 줄) 구조체 struct의 모든 배열 원소를 각 각 출력

# 4장 함수와 포인터 활용

## A. 함수의 인자전달 방식

### Ⅰ.값에 의한 호출과 참조에 의한 호출

c언어는 함수의 인자 전달 방식이 기본적으로 값에 의한 호출 방식이며 여기서 값에 의한 호출 방식은 함수 호출 시 실인자의 값이 형식인자에 복사되어 저장된다는 의미를 띄운다.

```
2   #include <stdio.h>
3
4   void increase(int origin, int increment);
5
6   int main(void)
7   {
8       int amount = 10;
9       //amount가 20 증가하지 않음
0       increase(amount, 20);
1       printf("%d₩n", amount);
2
3       return 0;
4   }
5
6   void increase(int origin, int increment)
7   {
8       origin += increment;
9   }
```

4번째 줄) 함수원형의 매개변수(int origin, int increment)로 정의하여 값에 의한 호출을 구현한다.

8번째 줄) 지역변수 amount를 선언하고 10을 저장

10줄) 함수호출 increase로 하여 함수 함수 increase()에서 변수 origin에 10이 각 increment에 20씩 저장

11번째 줄) amount 출력하면 10이 출력되게 구현

18번째줄) 매개변수 origin에 매개변수인 increment 값을 저장하고 main()의 11 행에서 10을 출력

### Ⅰ-Ⅱ.참조에 의한 호출

c언어에서 포인터를 매개변수로 사용하면 함수로 전달된 실인자의 주소를 이용하여 그 변수를 참조 할 수있는데 함수에서 주소의 호출을 **참조에 의한 호출** 이라한다.

```
1   #include <stdio.h>
2
3   void increase(int *origin, int increment);
4
5   int main(void)
6   {
7       int amount = 10;
8
9       increase(&amount, 20);
0       printf("%d₩n", amount);
1
2       return 0;
3   }
4
5   void increase(int *origin, int increment)
6   {
7
8       *origin += increment;
9   }
```

3번째 줄) 함수원형의 매개변수를 (int *origin, int increment)로 정의하여 origin을 참조에 의한 호출 구현

7번째 줄) 지역변수 amount를 선언하고 10을 저장

9줄) 함수호출 increase로 하여 함수 함수 increase()에서 변수 origin amount 주소를 저장하고 변수 increment 20씩 저장

10번째 줄) amount 출력하면 10이 출력되게 구현

18번째줄) main()의 변수 amount를 참조하여 매개변수인 increment의 값을 더하여 amount에 저장

## Ⅱ.배열의 전달

배열의 전달은 함수의 매개변수로 배열을 전달하는 것은 배열의 첫 워소를 참조 매개변수로 전달하는 것과 동일하다.

```
double sum(double ary[], int n);

double data[] = { 2.3, 3.4, 4.5, 6.7, 9.2 };

sum(data,5);
```

```
double sum(double ary[], int n)
{
    int i = 0;
    double total = 0.0;
    for (i = 0; i < n; i++)
        total += ary[i];

    return total;
}
```

함수sum()을 구현하는데 함수sum()은 실수형 배열의 모든 원소의 합을 구하여 반환하는 함수이다 sum()의 형식매개변수는 실수형 배열과 배열크기로 한다. 그러나 함수 내부에서 실인자로 전달된 배열크기는알 수 없고 매개변수 double ary[]처럼 배열형태로 기술해도 double*ary처럼 포인터 변수로인식하기 때문에 배열크기를 두 번째 인자로 사용한다.
배열의 크기를 인자로 사용하지 않으면 정해진 상수를 함수정의 내부에서 사용하는데 배열의 크기가 변하면 내용을 수정해야 하므로 비효율적이므로 배열크기에 관계없이 배열 원소의 합을 구하는 함수를 만드려면 배열크기도 하나의 인자로 사용하여야한다.

```
1  #include <stdio.h>
2
3  #define ARYSIZE 5
4  double sum(double g[], int n);
5
6  int main(void)
7  {
8
9      double data[] = { 2.3, 3.4, 4.5, 6.7, 9.2 };
10
11
12     for (int i = 0; i < ARYSIZE; i++)
13         printf("%5.1f", data[i]);
14     puts("");
15
16
17     printf("합: %5.1f₩n", sum(data, ARYSIZE));
18
19     return 0;
20 }
21
22
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 }
31
```

4번째 줄) 함수원형으로 double sum(double *, int n);도 가능하며 배열은 매개변수 포인터와 동일하다.

17번째 줄) 함수의 배열 매개변수에서 실인자를 호출할 때 이름을 data를 사용하고 sum ()으로 호출하면 함수 결과인 모든 배열원소의 합이 출력되도록 구현

23줄) 함수헤더 double sum()로 가능한데 배열은 매개변수에서 포인터와 동일하다.

27번째줄) 매개변수 ary는 double 포인터 형이여서 ary[i]로 (i+1)주소값을 참조 가능하여 main() 함수의 배열 data를 참조하여 모든 결과가 total에 저장되어 구현된다

29줄) return total을 사용해 total로 반환되게 구현

## Ⅲ. 가변인자

### Ⅲ-Ⅰ 함수 printf()의 함수원형 함수호출

```
int printf(const char* _Format, ...);

printf("%d%d%f", 3, 4, 3.678);
printf("%d%d%f%f%f", 7, 9, 2.45, 3.678, 8.98);
```

가변 인자가 있는 함수머리 printf()는 함수원형으로 되어 있다 첫 인자 char*_Format을 제외하고는 이후에 ...로 구현되어있다. 함수 printf()를 호출할때에는 출력할 인자의 수와 자료형이 결정되지 않은 채 호출하는데 출력할 인자의 수와 자료형은 인자_Format %d으로 표현되어있다.

### Ⅲ-Ⅱ 함수헤더에서의 가변인자 표현

함수에서 처음 또는 앞 부분의 매개변수는 정해져 있으나 이후 매개변수와 각각의 자료형이 고정적이지 않고 변하는 인작 함수의 가변인자 인데 매개변수에서 중간 이후부터 마지막에 위치한 가변인자만 가능하며 함수를 정의 시 가변인자 매개변수는 ...으로 기술하고 함수 vast의 함수 헤드는 void vast(int n...)와 같이 가변인자인 ...의 앞 부분에는 매개변수가 int n처럼 고정적이여야하고 이후에 가변인자인 ...을 구현 할수 있다.

```
int vatest(int n, ...);
double vasum(char* type, int n, ...);
double vafun1(char* type, ..., int n);
double vafun2(...);
```

### Ⅲ-Ⅲ 가변 인자가 있는 함수 구현

함수에서 가변 인자를 구현하려면 **가변인자 선언**, **가변인자 처리 시작**, **가변인자 얻기**, **가변인자 처리 종료** 4단계를 필요로하는데 구현하기 위해서는 헤더파일 stdarg.h가 필요하다.
**가변인자 선언**은 변수선언처럼 가변인자로 처리할 변수를 하나 만드는 일로서 자료형인 **va_list**는 가변 인자를 위한 char*로 헤더파일 stdarg.h가 정의되어있다.

**가변인자 처리시작은** 선언된 변수에서 마지막 고정 인자를 지정해 가변 인자의 시작 위치를 알리는 방법으로 함수 **va_start()** 헤더파일 stdarg.h에 정의되어 있는 매크로 함수이다.

**가변인자 얻기 처리**는 가변인자 각각의 자료형을 지정하여 가변인자를 반환 받는 절차로 함수 va_arg()도 헤더파일 stdarg.h에 정의된 매크로 함수이다. 매크로 함수 **va_arg()**의 호출로 반환된 인자로 원하는 연산을 처리할 수 있다.

**가변인자 처리 종료**는 표현 그대로 가변 인자에 대한 처리를 끝내는 단계로 **va_end()** 함수는 헤더파일 stdarg.h에 정의된 매크로 함수이다.

| 구문 | 처리 절차 | 설명 |
|---|---|---|
| va_list argp; | 1.가변인자 선언 | va_list로 변수 argp를 선언 |
| va_start(va_list argp) | 2.가변인자 처리 시작 | va_start()는 첫 번째 인자로 va_list로 선언된 변수이름 argp 두 번째 인자는 가변인자 앞의 고정인자 prearg를 지정 |
| type va_arg(va_list atgp) | 3.가변인자 얻기 | va_arg()는 첫 번째 인자로va_start()초기화한 va_list변수 argp를 받고 type을 기술 |
| va_end(va_list argp) | 4.가변인자 처리 종료 | va_list로 선언된 변수이름 argp의 가변인자 처리 종료 |

**가변인자 선언**(va_list 가변인자변수)
**가변인자 처리시작**(va_start 가변인자변수, 가변인자 첫 고정인자)
**가변인자 얻기(**va_arg가변인자변수 반환될 자료형)
**가변인자 처리종료**(va_end가변인자변수)

```
1
2   #include <stdio.h>
3   #include <stdarg.h>
4
5   double avg(int count, ...);
6
7   int main(void)
8   {
9       printf("평균 %.2f₩n", avg(5, 1.2, 2.1, 3.6, 4.3, 5.8));
10
11      return 0;
12  }
13
14
15  double avg(int numagrs, ...)
16  {
17
18      va_list argp;
19
20
21      va_start(argp, numagrs);
22
23      double total = 0;
24      for (int i = 0; i < numagrs; i++)
25
26          total += va_arg(argp, double);
27
28
29      va_end(argp);
30
31      return total / numagrs;
32  }
```

3번째 줄) 가변인자 처리를 위한 자료형과 매크로가 정의되어 있는 헤더파일 stdarg.h 삽입

5번째 줄) 가변인자 처리 함수원형으로 double avg에서 ...을 가변인자로 표시

15줄) 가변인자 처리 함수 avg()의 함수헤더

18번째줄) 가변인자 변수 argp를 자료형 va_list를 선언 헤더파일 stdarg.h에 정의됨

21줄) numargs 이후가 가변인자의 시작임을 알리기 위해 사용하고 va_start를 호출

26줄) 가변인자를 얻기 위해 가변인자 각각의 자료형을 지정하고 va_arg로 호출

26줄) 가변인자 종료를 알리는 매크로 va_end 호출

B. 포인터 전달과 반환

## Ⅰ.매개변수와 반환으로 포인터 사용

### Ⅰ-Ⅰ 주소연산자 &

함수에서 매개변수를 포인터로 이용하면 참조에 의한 호출이된다.

```
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;
}
```

함수원형 void add(int *, int,int);에서 첫 매개변수는 포인터인 int*이며 함수add()는 두 번째와 세 번째 인자를 합하여 첫 번째 인자가 가리키는 변수에 저장하는 함수이다.
함수들을 호출하는 방법은 합이 저장될 변수인 sum을 선언하여 주소값인 &sum을 인자로 호출 한다.

### Ⅰ-Ⅱ 주소값 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;

}
```

함수의 결과를 포인터로 변환하는데 함수원형을 int * add(int *, int,int)로 하는 함수 add()는 반환값이 포인터인 int*이며 함수 add()정의에서 두 수의 합을 첫 번째 인자가 가리키는 변수에 저장한 후 포인터인 첫 번째 인자를 그대로 반환한다. 함수 add()를 *add(&sum, m,n)호출하면 변수 sum에 합 a+b가 저장되며 반환값인 포인터가 가리키는 변수인 sum을 바로 참조할 수 있다.

## Ⅰ-Ⅲ 키워드 const

포인터를 매개변수로 이용하면 수정된 결과를 받을 수 있어 편리한 점은 존재하나 참조에 의한 호출은 매개변수가 가리키는 변수값이 원하지 않는 값으로 수정될 수 있다. 이러한 포인터 인자의 잘못된 수정을 미리 예방하는 방법이 있는데 수정을 원하지 않는 함수의 인자 앞에 키워드 const를 삽입하여 참조되는 변수가 수정될 수 없게 한다.

```
void multiply(double* result, const double* a, const double* b)
{
    *result = *a * *b;

    *a = *a + 1;
    *b = *b + 1;

}
```

키워드 const는 인자인 포인터 변수가 가리키는 내용을 수정할 수 없도록 하고 인자를 const double *a와 const double *b로 구현하면 *a *b를 대입연산자의 1-value로 사용할 수 없다 상수 키워드 const의 위치는 자료형 앞이나 포인터변수 *a앞에도 가능하므로 const double *a와 double const *a는 동일한 표현으로 간주된다.

## Ⅰ-Ⅳ 복소수를 위한 구조체

복소수의 자료형인 구조체 complex는 실수부와 허수부를 나타내는 real과 img를 멤버로 구성이된다.

```
struct complex
{
    double real;
    double img;

};

typedef struct complex complex;
```

자료형 struct complex와 자료형 complex를 선언하고 둘 모두 복소수 자료형으로 사용이 가능하다

## Ⅰ-Ⅴ 인자와 반환형으로 구조체 사용

구조체는 함수의 인자와 반환값으로 이용이 가능하다 함수는 구조체 인자를 값에 의한 호출 방식을 이용하고 함수 paircomplex1() 내부에서 구조체 지역변수 com을 하나 만들어 실인자의 구조체 값을 모두 복사하는 방식으로 구조체 값을 전달 받고 구조체 자체를 전달하기 위해 대입하고 다시 반환값을 대입하려면 시간이 소요가된다. 이 함수는 유형이 complext인 인자의 켤레 복소수를 구하여 그 결과를 반환한다.함수에서 구조체의 반환 방법도 다른 일반 변수와 같다.

C. 함수 포인터와 void 포인터

## Ⅰ.함수 포인터

### Ⅰ-Ⅰ 함수 주소 저장 변수

먼저 함수 포인터의 장점은 다른 변수를 참조하여 읽거나 쓰는것도 가능하며. 함수의 주소값을 저장하는 포인터변수 이다. 이처럼 하나의 함수 이름으로 필요에 따라 여러 함수를 사용하면 편리하게 만들어주는게 함수 포인터이다. 함수 포인터는 반환형, 인자목록의 수와 각각의 자료형이 일치하는 함수의 주소를 저장할 수 있고 포인터를 선언하려면 함수원형에서 함수이름을 제외한 반환형과 인자목록의 정보가 필요로한다.

```
void add(double*, double, double);
void subtrcat(double*, double, double);

...

void (*pf1)(double* z, double x, double y) = add;
void (*pf2)(double* z, double x, double y) = subtract;

pf2 =  add;
```

변수이름 pf 함수 포인터를 선언한 뒤 함수 포인터 pf를 void add(double*, double,double); 인 함수를 저장하기위해 함수원형에서 반환형인 void와 인자목록인(double*, double, double)을 필요로한다. 함수 포인터 pf는 문장 void(*pf)(double*, double,double);문장으로 선언될 수도 있다.

```
void (*pf2) (double* z, double x, double y) = add();
pf2 = subtract();
pf2 = add;
pf2 = &add;
pf2 = subtract;
pf2 = &subtract;
```

포인터 변수 pf는 함수 add()만을 가리킬 수있는 것이 아니라 add()와 반환형과 인자목록이 같은 함수는 모두 가리킬 수있다 subtract()의 반환형과 인자목록이 add()와 동일하다면 pf는 subtract()도 가리킬 수 있다. 문장 pf = subtract;와 같이 함수 포인터에는 괄호가 없이 함수이름으로만 대입해야 한다. 함수 이름 add나 subtract는 주소 연산자를 함께 사용하여 &add나 &subtract로 사용가능하고 대신에 subtract()와 add()와 같이 함수호출로 대입해서는 오류가 발생하여 다른 방법으로 구현해야한다.

## Ⅰ-Ⅱ 함수 포인터 배열

포인터 배열과 같이 함수 포인터가 배열의 우너소로 여러 개의 함수 포인터를 선언하는 함수 포인터 배열을 사용한다. 함수 포인터 배열은 함수 포인터가 원소인 배열이다.

```
void add(double*, double, double);
void subtract(double*, double, double);
void multiply(double*, double, double);
void devide(double*, double, double);
    ...
    void (*fpaty[4]) (double*, double, double);
```

반환자료형을 선언하고 배열이름 크기와 자료형1 매개변수이름1 씩 선언한 뒤 배열 fpaty의 각 원소가 가리키는 함수는 반환값이 void이고 인자목록이(double*, double, double)인 경우의 배열 선언을 하는 과정이다.

```
void (*fpary[4]) (double*, double, double);
fpary[0] = add;
fpary[1] = subtract;
fpary[2] = multipy;
fpary[3] = devide;

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

함수 포인터 배열 선언과 초기화는 fpary을 선언한 이후에 함수 4개를 각각의 배열원소에 저장하는 방법과 배열 fpary을 선언하면서 함수 4개의 주소값을 초기화하는 문장을 구현하뒤 배열의 선언과 초기화 문장으로 처리한다.

## Ⅰ-Ⅲ void 포인터

void포인터는 자료형을 무시하고 주소값만을 다루는 포인터인데 void포인터는 대상에 상관없이 모든 자료형의 주소를 저장할 수 있는 만능 포인터로도 사용가능하고 void 포인터에는 일반 변수 포인터는 물론 배열과 구조체 담을 수있다.

```
1
2     #include <stdio.h>
3
4     void myprint(void)
5     {
6         printf("void 포인터, 신기하네요!₩n");
7     }
8
9     int main(void)
10    {
11        int m = 10;
12        double d = 3.98;
13
14        void *p = &m;
15        printf("주소 %d₩n", p);
16        //printf("%d₩n", *p);
17
18        p = &d;
19        printf("주소 %d₩n", p);
20
21        p = myprint;
22        printf("주소 %d₩n", p);
23
24        return 0;
25    }
```

14번째 줄) void 포인터 p를 선언하여 초기값으로 int 포인터인 m의 주소를 저장

15번째 줄) int 포인터인 m의 주소 출력

18번째 줄) void 포인터 p에 double 포인터인 d의 주소를 저장

21번째 줄) void 포인터 p에 함수 myprint()의 주소를 저장

22번째 줄) double 포인터인 d의 주소 출력

# 포트폴리오를 작성하면서 느낀점

군 복무를 마치고 1년 동안 돈 벌다가 다시 학업을 진행하게 되었다. 3년 동안의 공백이 쉽사리 채워지지 않을 거라는 생각을 하고 학교를 다니면서 열심히 배우려고 했으나 때마침 코로나 사태가 날이 갈수록 심해지는 추세여서 비대면 수업 방식으로 진행하는 바람에 교수님들과 직접 소통도 어렵고 즉시 모르는 문제들을 질문할 수도 없는 터라 어려움을 겪고 있는데 마침 중간고사 과제가 1학기 중간고사 시험 전까지 배운 내용들을 스스로 정리하고 검토하여 포트폴리오로 작성하는 것인데 이 시간이 너무 감사하고 복습하는 시간이 되어서 뜻깊은 시간이였고 작성하면서 어려운 부분들도 많았지만 스스로 터득하고 알아갈 수 있어서 보람찬 기회였다.