

# <C언어 포트폴리오>

반: 2-E

학번: 20191796

이름: 류상배

깃허브 주소:

<https://github.com/fbtkdgo/2020-C-Code>

## <목차>

### 1. 머리말

### 2. C 언어 소개

- C언어란?

### 3. 1학년 2학기 C언어 학습과정

- 프로그래밍언어의 개요
- C프로그래밍 첫걸음
- 자료형과 변수 (★★)
- 전처리와 입출력 (★)
- 연산자 (★)
- 조건문 (★)
- 반복문 (★)
- 포인터의 기초 (★★★)
- 배열 (★★)
- 함수의 기초 (★)

### 4. 2학년에 들어와 지금까지의 학습과정

- 문자와 문자열 (★)
- 변수 유효범위
- 구조체와 공용체 (★★)
- 함수와 포인터 활용 (★★★)
- 파일 처리~ (★)

## 1. 머리말

작년에 공부를 소홀히 하기도 했고 아직 개념이 잡히지 않은 C언어는 물론 전체적인 프로그래밍에 대한 기초가 부족한 느낌이어서 이번에 포트폴리오 작성을 기회 삼아 공부했던 내용들을 제 방식으로 다시 정리해보는 시간을 가졌습니다.

## 2. C언어란?

C 언어는 운영체제를 만드는 프로그래밍 언어로 널리 알려져 있다. 모든 운영체제의 기본이 되는 유닉스는 70년대 초 C언어를 기반으로 개발되었다. 지금 널리 알려진 마이크로소프트사의 윈도우즈, 맥킨토시의 OS X 그리고 리눅스 또한 C 언어로 개발되었다. C 언어는 운영체제 뿐만 아니라 임베디드 시스템과 다양한 분야의 응용 프로그램 개발에도 사용되고 있다. 이렇게 많은 프로그램이 여러 분야에 퍼져 있어 현재에도 C 언어는 학습과 실무에서 중요한 고급언어의 기본 언어로 자리매김 되어있다. C 언어를 배움으로서 C++, C#, JAVA, Python 등 대부분의 언어의 이해와 학습에 큰 도움이 된다.

## 3. 프로그래밍언어의 개요

### (1) '프로그램'이 무엇일까?

- 스마트폰과 노트북, 혹은 데스크탑 컴퓨터에서 특정 작업을 위해 컴퓨터를 작동 시키는 것.
- 컴퓨터와 스마트폰에서 특정 목적의 작업을 수행하기 위한 관련 파일의 모임.
- 프로그램은 특정 작업을 수행하기 위하여 그 처리 방법과 순서를 기술한 명령어와 자료로 구성되어 있고, 컴퓨터에게 지시할 일련의 처리 작업 내용을 담아 사용자의 프로그램 조작을 더해 적절한 명령을 지시하여 실행됨.

#### 프로그래머

- 컴퓨터와 스마트폰 등의 정보기기에서 사용되는 프로그램을 만드는 사람을 뜻함.

#### 프로그래밍 언어

- 사람과 컴퓨터가 서로 의사 교환을 하기 위한 언어.
- 프로그램을 개발하기 위해 사용하는 언어.
- 사람이 컴퓨터에게 지시할 명령어를 기술하기 위하여 만들어진 언어.

- FORTRAN, ALGOL, BASIC, COBOL, PASCAL, C, C++, Visual Basic, Delphi, Java, Objective-C, Perl, JSP, Javascript, Python, C#, Go 등이 있다.

## (2) 언어의 계층과 번역

- 기본적으로 컴퓨터는 하드웨어와 이를 작동하게하는 소프트웨어로 구성.
- 하드웨어의 구성요소는 대표적으로 CPU, 주기억장치, 보조기억장치, 입력장치, 출력장치로 이루어져 있다.
- 소프트웨어는 컴퓨터가 수행할 작업을 지시하는 전자적 명령어들의 집합으로 구성된다.
- 소프트웨어는 크게 응용 소프트웨어와 시스템 소프트웨어로 나눈다.
- 운영체제는 컴퓨터 하드웨어 장치의 전반적인 작동을 제어하고 조정하며, 사용자가 최대한 컴퓨터를 효율적으로 사용할 수 있도록 시스템 프로그램
- 컴퓨터는 전기의 흐름을 표현하는 1과 흐르지 않는 0으로 표현되는 기계어만을 인식한다. 그러므로 사용자가 컴퓨터에게 명령을 내리기 위해서는 프로그래밍언어를 기계어로 변환해주는 컴파일러가 필요하다.
- 어셈블리어는 기계어를 사용자가 더 이해하기 쉬운 기호 형태로 일대일 대응시킨 언어이다. LDA, ADD, STA 등이 있으며 명령어를 기호화한 것을 니모닉이라 한다. 어셈블리어는 기계어로 변환해주는 어셈블러가 필요하다.
- 기계어와 어셈블리어는 저급언어, 우리가 배우는 C, JAVA 등은 고급언어이다.

## (3) 프로그래밍의 자료 표현

- 0~9의 한 자리수에 사용하는 숫자를 십진수라 하고, 기계어처럼 on과 off만을 표현하는 0과 1의 숫자를 이진수라 한다.
- 정보 처리 단위중 가장 작은 기본 정보단위는 bit이다.
  - \* 비트가 연속적으로 8개 모인 정보단위를 바이트
  - \* 바이트가 4개, 8개 모이면 워드라 한다. (시스템마다 다름)
- 참과 거짓을 의미하는 두 가지 정보를 논리값이라 한다.
- 컴퓨터에서 문자를 표현할 때 개인용에서 가장 많이 쓰이는 것은 'ASCII Code'이고, 전 세계의 문자를 일관되게 표현하려고 만든 것은 'Unicode'이다.

## 4. C프로그래밍 첫걸음

### (1) 프로그램 구현 과정과 통합개발환경

- 소프트웨어 개발의 5단계 요구분석, 설계, 구현, 구현, 검증, 유지보수의 단계를 거친다.
- 프로그램 구현의 5단계 구상, 소스편집, 컴파일, 링크, 실행의 단계를 거친다.
- C언어로 명령어를 저장한 것을 소스파일이라고하고 텍스트파일로 저장된다.
- \* 소스파일은 프로그래밍 언어에 따라 고유한 확장자를 가짐.
- 프로그램 개발에 필요한 편집기, 컴파일러, 링커, 디버거를 통합하여 편리하고 효율적으로 제공하는 개발환경을 통합개발환경(IDE)라 한다.

## (2) C프로그램의 이해와 디버깅 과정

- C프로그램과 같은 절차지향 프로그램은 함수로 구성된다. 프로그래머가 직접 만드는 함수를 사용자 정의 함수라 칭하고, 시스템에 만들어져 있는 함수를 라이브러리 함수라 한다.
- 디버깅은 오류가 발생할만한 지점을 중단점으로 설정하는 것을 시작으로 해 컴파일 오류, 링크 오류, 논리 오류를 수정하는 것이다.

## 5. 자료형과 변수

### (1) 프로그래밍 기초

- 프로그램 코드를 작성하는 사람이 해당하는 단어를 다른 용도로 사용하지 못하는 것을 예약어 즉 키워드라고 한다.
- 프로그래머가 자기 마음대로 정의해서 사용하는 것을 식별자라고 하는데 이는 키워드와 철자, 대문자, 소문자 등을 변경하여 겹치지 않아야 하고 공백이 들어갈 수 없다.
- 컴퓨터에게 명령을 내리는 최소 단위를 문장이라고 하고 보통 세미콜론(;)으로 종료된다. 이 문장들이 여러개 묶으면 블록이 된다.
- 주석이란 프로그램 동작에 아무런 영향을 끼치지 않지만 프로그램을 짜는 과정에서 전체적인 이해력을 돕는데 도움을 주므로 중요한 역할을 한다.

### (2) 자료형과 변수선언

- 자료형이란 프로그래밍 언어에서 자료를 식별하는 종류를 말한다.
- 정수와 실수, 문자 등의 자료값을 중간 중간에 저장할 공간을 변수라 부르는데 고유한 이름을 갖고 물리적인 기억장치인 메모리에 위치한다. 선언된 자료형에

따라 저장공간 크기와 자료값의 종류가 결정된다.

- 변수선언은 컴파일러에게 프로그램에서 사용할 저장 공간인 변수를 알리는 역할이며, 프로그래머 자신에게도 선언한 변수를 사용하겠다는 약속의 의미가 있다. 변수를 사용하기 위해서는 원칙적으로 사용 전에 변수선언을 해야한다.
- 변수초기화란 변수를 선언한 후 값을 저장하는 것이고 저장을 하지 않으면 원하지 않는 값이 저장되며, 오류가 발생한다.

### (3) 기본 자료형

- integer는 키워드로 int이며 정수형 자료형이다. 여기서 파생된 자료형은 long과 short가 있다. short는 int보다 작거나 같고, long은 int보다 크거나 같다.
- signed 자료형과 unsigned 자료형은 정수형 앞에 사용되는데 signed 자료형은 부호가 있는 자료형을 뜻해 사용할 때 생략이 가능하지만 unsigned 자료형은 부호가 없어서 0과 양수만을 저장할 수 있다.
- 부동소수형 자료형의 키워드는 float, double, long double 세 가지가 있다. float형보다 double형이 정확하고 double형보다 longdouble이 정확하다.  
\*float형 변수에 저장하면 꼭 5.28F와 같이 상수로 저장해야 한다.
- 문자형 자료형의 키워드는 char, signed char, unsigned char 세 가지가 있다. 문자형 저장공간의 크기는 모두 1바이트로 동일하다.
- 자료형의 크기와 값의 범위 (이미지 첨부)

자료형	크기	값의 범위
char	1 바이트	-128~127, 0~255
unsigned char	1 바이트	0~255
signed char	1 바이트	-128~127
int	2 또는 4 바이트	-32,768 ~ 32,767 또는 -2,147,483,648 ~ 2,147,483,647
unsigned int	2 또는 4 바이트	0 ~ 65,535 또는 0 ~ 4,294,967,295
short	2 바이트	-32,768 ~ 32,767
unsigned short	2 바이트	0 ~ 65,535
long	4 바이트	-2,147,483,648 ~ 2,147,483,647
unsigned long	4 바이트	0 ~ 4,294,967,295

- 자료형의 범주에서 벗어난 값을 저장하면 초과시에 오버플로우, 매우 자세한 숫자를 적을시에 언더플로가 발생한다.

#### (4) 상수 표현방법

- 상수는 이름 없이 있는 그대로 표현한 자료값이나 이름이 있으나 정해진 하나 값만으로 사용되는 자료값을 뜻한다. 리터럴 상수와 심볼릭 상수로 구분이 가능.
- 리터럴 상수는 소스에 그대로 표현에 의미가 전달되는 다양한 자료값을 뜻하고 심볼릭 상수는 리터럴 함수와 달리 이름을 갖는 상수를 뜻한다.
- 심볼릭 상수는 const 상수, macro 상수, 열거형 상수로 표현된다.
- 일반적으로 상수의 정수표현은 십진수로 인식되나 숫자 0을 정수 앞에 놓으면 팔진수(0~8), 0x를 숫자앞에 놓으면 십육진수(0~f)로 인식한다. 함수 printf()에서 정수형을 출력할 때 %d를 사용하는데 이는 십진수의 demical의 약자이다.
- 지수표현 방식은 e또는 E를 사용하는데 예시로 3.14E+2는  $3.14 \times 10^2$ 이다.
- 열거형 상수는 키워드 enum을 사용하여 정수형 상수 목록 집합을 정의한다. 목록 첫 상수의 기록값이 0이며 다음부터 1씩 증가하는 방식으로 상수값이 자동 부여 된다.
- 전처리 지시자 #define은 매크로 상수를 정의한다.

## 6. 전처리와 입출력

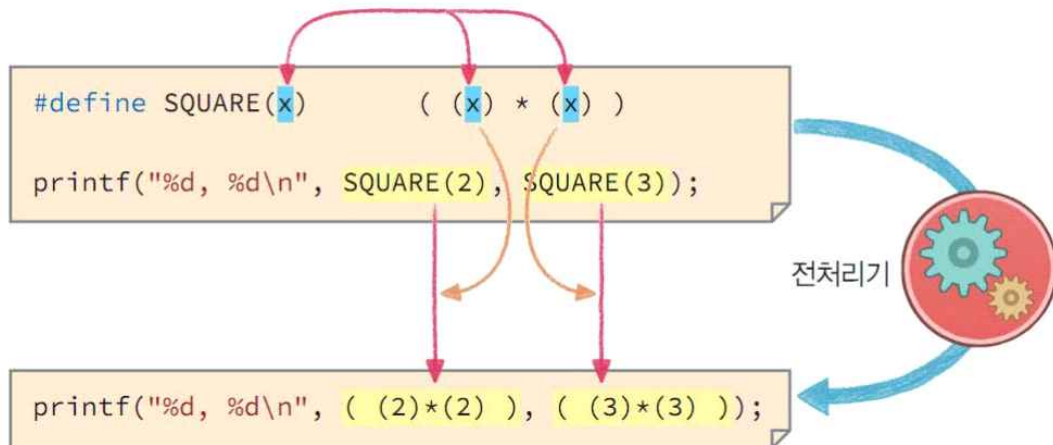
### (1)전처리

- C언어는 컴파일러가 컴파일하기 전 전처리기의 전처리 과정이 필요하다. 전처리 과정에서 처리되는 문장을 전처리 지시자라 한다. #define, #include가 해당
- #include는 프로그램에서 함수를 사용하기 위한 헤더파일을 삽입할 때 사용된다.
- 주요헤더파일

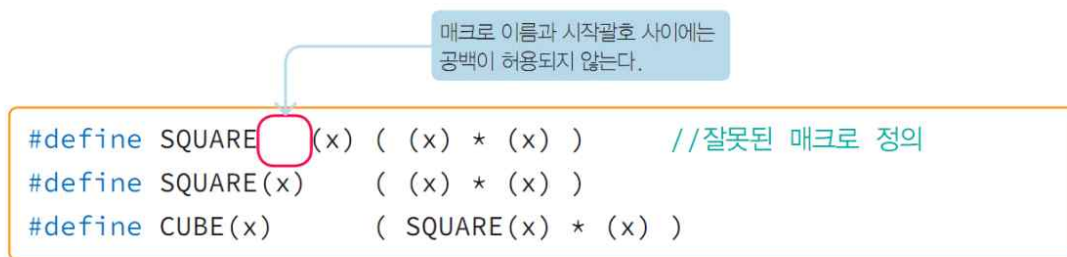
헤더파일	파일이름	파일내용
stdio.h	STanDard Input Output	표준입출력 함수와 상수
stdlib.h	STanDard LiBrary	주요메모리 할당 함수와 상수
math.h	math	수학 관련 함수와 상수
string.h	string	문자열 관련 함수와 상수
time.h	time	시간 관련 함수와 상수
ctype.h	Character Type	문자 관련 함수와 상수
limits.h	limits	정수와 상수 등 여러 상수
float.h	float.h	부동소수에 관련된 각종 상수

-인자를 사용한 매크로

인자인 x부분이 실제 사용한 수로 대체된다.



\* 주의할 점



## (2) 출력 함수 printf()

- printf()의 인자는 크게 형식문자열과 출력할 목록으로 구분하고, 출력 목록의 각 항목을 형식문자열에서 %d와 같이 %fh 시작하는 형식지정자 순서대로 서식화하여 그 위치에 출력한다. 첫 번째 인자인 형식문자열은 일반 문자와 이스케이프 문자, 그리고 형식 지정자로 구성된다.
- 형식지정자는 출력 내용의 자료형에 따라 %d, %i, %c, %s 등이 있다.
- 정수를 출력할 때 십진수 형식은 %d, %i 팔진수 형식은 %o 십육진수 형식은 %x를 사용한다. 이 때 팔진수와 십육진수의 원래 형태를 출력하고 싶다면 #을 삽입하여 출력하면 된다.
- 출력 필드 폭이 출력 내용의 폭보다 넓으면 정렬은 기본이 오른쪽이면 필요하면 왼쪽으로 지정이 가능하다.



- 형식 문자의 종류

서식 지정자	출력 형태	서식 지정자	출력 형태
%c	단일 문자	%x	부호없는 16진정수(소문자)
%d	부호 있는 10진 정수	%X	부호없는 16진정수(대문자)
%i	%d와 같음	%e	e표기법에 의한 실수
%f	부호 있는 10진 실수	%E	E표기법에 의한 실수
%s	문자열	%g	값에 따라서%f,%e중 하나를 선택
%o	부호 없는 8진 정수	%G	값에 따라서%f,%e중 하나를 선택
%u	부호 없는 10진 정수	%%	%기호 출력

- 옵션지정 문자의 종류

문자	기본(없으면)	의미	예와 설명
-	우측정렬	수는 지정된 폭에서 좌측정렬	%-10d
+	음수일 때만 - 표시	결과가 부호가 있는 수이면 부호 +, -를 표시	%+10d
0	0을 안 채움	우측정렬인 경우, 폭이 남으면 수 앞을 모두 0으로 채움	%010x %-0처럼 좌측정렬과 0 채움은 함께 기술해도 의미가 없음
#	리딩 문자 0, 0x, 0X가 없음	서식문자가 o(서식문자 octal)인 경우 0이 앞에 붙고, x(서식문자 hexa)인 경우 0x가 붙으며, x인 경우 0X가 앞에 붙음	수에 앞에 붙는 0이나 0x는 0으로 채워지는 앞 부분에 출력

- 정수의 다양한 출력

형식지정자	정수	결과
%d	1234	1234
%6i	1234	1234
%+6i	1234	+1234
%-06i	1234	+01234
%60	037	037
%-60	037	37

형식지정자	정수	결과
%-#6o	037	037
%#-6o	037	037
%05x	0x1f	0001f
%0#6x	0x1f	0x001f
%-0#5x	0x1f	0x1f
%#6X	0x1f	0X1F

- 문자열의 다양한 출력

형식지정자	인자(문자열)	결과
%5.2s	"Hello!"	He
%7.2s	"Hello!"	He
%-8.6s	"Hello!"	Hello!
%-8.3s	"Hello!"	Hel

형식지정자	인자(문자열)	결과
"%5.*s", 3	"Hello!"	Hel
"%*.4s", 7	"Hello!"	Hell
%*.*s, 8, 6	"Hello!"	Hello!
%*.*s, -8, 6	"Hello!"	Hello!

### (3) 입력 함수 scanf()

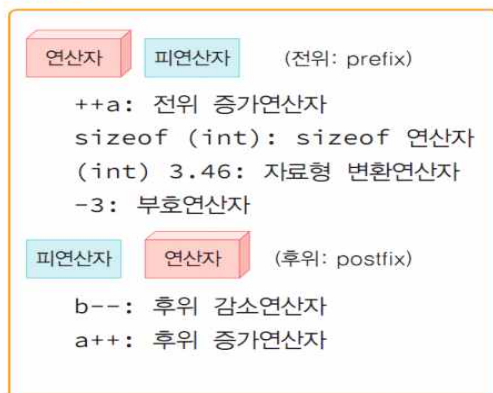
- scanf()는 대표적인 입력 함수로 형식지정자는 printf()와 동일하다.
- scanf()의 첫 번째 인자는 형식문자열이라 하며 형식지정자와 일반문자로 구성되고, 두 번째 인자부터는 키보드 입력값이 복사 저장되는 입력변수 목록으로 변수 이름 앞에 반드시 주소연산자&를 붙여 나열한다. &는 주소연산자로 뒤에 표시된 피연산자인 변수 주소값이 연산값으로 scanf()의 입력변수목록에는 키보드에 입력값이 저장되는 변수를 찾는다는 의미에서 반드시 변수의 주소연산식 '& 변수이름'이 인자로 사용되어야 한다.
- 실수의 입력 scanf()에서 자료형이 float이면 %f, double이면 %lf로 구분하여 사용한다.
- 함수 getchar()는 문자 하나를 입력하는 매크로 함수이고, putchar()는 반대로 출력하기 위한 매크로 함수이다.
- scanf()의 경고와 함께 등장한 scanf\_s()는 Visual C++ 2005 버전부터 보안 문제로 scanf()의 사용을 권장하지 않게되면서 보안이 강화된 버전이다.

## 7. 연산자

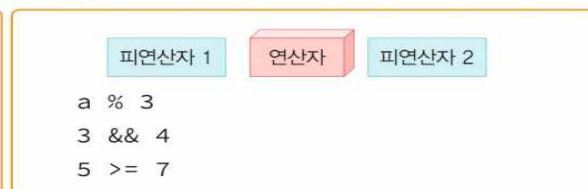
### (1) 연산식과 다양한 연산자

- 변수와 다양한 리터럴 상수 그리고 함수의 호출 등으로 구성되는 식을 연산식이라 한다. 연산식은 반드시 하나의 결과 값인 연산 값을 갖고, 산술연산자 +, -, \* 기호와 같이 이미 정의된 연산을 수행하는 문자 또는 문자조합 기호를 말한다. 그리고 연산에 참여하는 변수나 상수를 피연산자라고 한다.
- 연산식은 평가하여 항상 하나의 결과값을 갖고 '연산값'이라 부른다.
- 연산자는 연산에 참여하는 피연산자의 개수에 따라 단항, 이항, 삼항으로 나눌 수 있다.
- 단항, 이항, 삼항 연산자의 예시

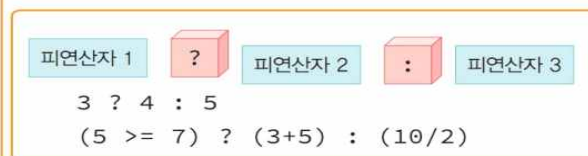
#### 단항연산자



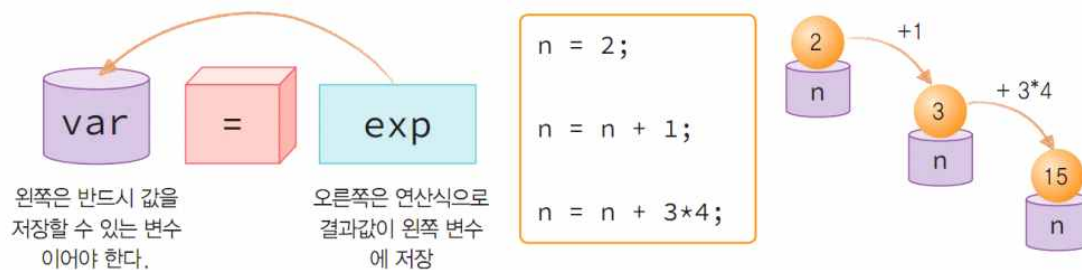
#### 이항연산자



#### 삼항연산자



- 산술연산자로는 +(더하기), -(빼기), \*(곱하기), /(나누기), %(나머지)가 있고, 부호연산자 +, - 는 숫자 앞에 붙어 양수와 음수를 표현한다. 대입연산자 =는 오른쪽의 연산 값을 변수에 저장하는 연산자이다.
- 대입 연산자의 예시



## (2) 관계와 논리, 조건과 비트연산자

- 관계연산자는 두 피연산자의 크기를 비교하기 위한 연산자이다. 관계연산자의 연산값은 비교 결과가 참이면 0, 거짓이면 1이다.

연산자	연산식	의미	예제	연산(결과)값
>	$x > y$	x가 y보다 큰가?	$3 > 5$	0(거짓이면)
>=	$x \geq y$	x가 y보다 크거나 같은가?	$5-4 \geq 0$	1(참이면)
<	$x < y$	x가 y보다 작은가?	'a' < 'b'	1(참이면)
<=	$x \leq y$	x가 y보다 작거나 같은가?	$3.43 \leq 5.862$	1(참이면)
!=	$x \neq y$	x와 y가 다른가?	$5-4 \neq 3/2$	0(거짓이면)
==	$x == y$	x가 y가 같은가?	'%' == 'A'	0(거짓이면)

- 논리연산자 &&, ||, !은 각각 and, or, not의 논리연산을 의미하며, 그 결과가 참이면 1 거짓이면 0을 반환한다. C언어에서 참과 거짓의 논리형은 따로 없으므로 0, 0.0, W0은 거짓을 의미하며, 0이 아닌 모든 정수와 실수, 그리고 널 문자 'W0'가 아닌 모든 문자와 문자열은 모두 참을 의미한다.

- 논리연산자의 예시

x	y	$x \&\& y$	$x \parallel y$	!x
0(거짓)	0(거짓)	0	0	1
0(거짓)	0(0이 아닌 값)	0	1	1
0(0이 아닌 값)	0(거짓)	0	1	0
0(0이 아닌 값)	0(0이 아닌 값)	1	1	0

```

21 && 3           1
!2 && 'a'         0
3>4 && 4>=2       0
1 || '\0'         1
2>=1 || 3 <=0     1
0.0 || 2-2        0
!0                1

```

- 조건연산자는 삼항연산자에서만 사용되며 조건에 따라 주어진 피연산자가 결과값이 된다.

- 조건연산자의 예시

```
max = (a > b) ? a : b;           //최대값 반환 조건연산
max = (a < b) ? b : a;           //최대값 반환 조건연산
min = (a > b) ? b : a;           //최소값 반환 조건연산
min = (a < b) ? a : b;           //최소값 반환 조건연산
absolute = (a > 0) ? a : -a;      //절대값 반환 조건연산
absolute = (a < 0) ? -a : a;      //절대값 반환 조건연산
```

- 비트 논리 연산자는 피연산자 정수값을 비트 단위로 논리 연산을 수행하는 연산자로 &, |, ^, ~ 4가지이다. 비트연산은 각 피연산자를 int형으로 변환하여 연산하며 결과도 int형이다.

- 비트 논리 연산자의 예시

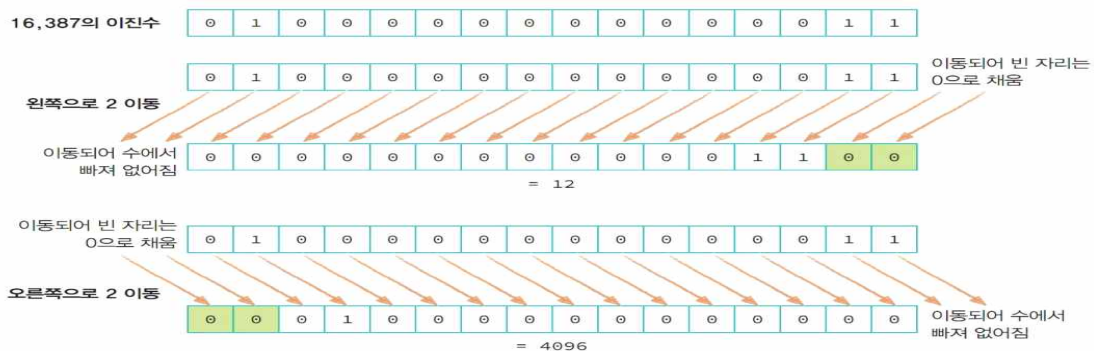
x(비트1)	y(비트2)	x & y	x   y	x ^ y	~x
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

- 비트 논리 연산에서 각 비트 연산 방법

x(비트1)	y(비트2)	x & y	x   y	x ^ y	~x
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

- 비트 이동연산자 >>, <<는 연산자의 방향인 왼쪽이나 오른쪽으로, 비트 단위로 줄줄이 이동시키는 연산자이다.

- 비트 이동 연산자의 예시



### (3) 형변환 연산자와 연산자 우선순위

- 자료형 char와 int는 각각 문자와 정수를 표현하고 각각 1바이트와 4바이트로 크기도 다르다. 이러한 char형과 int 형 간과 같이 필요에 따라 자료의 표현방식을 바꾸는 것을 자료형 변환이라 하고, 범주 변화에 따른 구분으로 올림변환과 내림변환으로 나눌 수 있다.

올림변환: 작은 범주의 자료형(int)에서 보다 큰 범주인 형(double)으로의 형변환

내림변환: 큰 범주의 자료형(double)에서 작은 범주인 형(int)로의 형변환

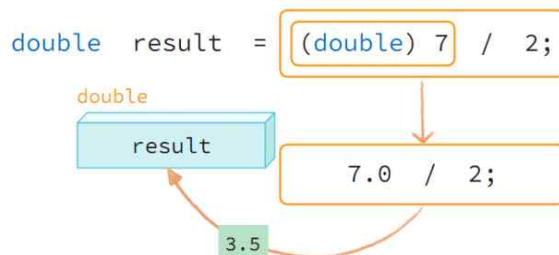
- 올림변환은 정보의 손실이 없어서 컴파일러에 의해 자동으로 수행될 수 있는데 이를 묵시적 형변환이라 하고, 내림변환은 묵시적 형변환을 할 경우 자료의 손실이 일어날 수 있어서 경고를 발생한다.

- 형변환 연산자 '(type)' 피연산자'는 뒤에 나오는 피연산자 값을 괄호에서 지정한 자료형으로 변환하는 연산자이다. 형변환 연산자를 사용한 방식을 명시적 형변환이라 한다.

- 상수나 변수의 정수값을 실수로 변환하려면 올림변환을 사용하고, 반대로 실수의 소수부분을 없애고 정수로 사용하려면 내림변환을 사용한다.

\* 단항연산자인 형변환 연산자는 모든 이항연산자보다 먼저 계산한다.

- 형변환 연산자와 산술연산자의 예시



#### 다양한 형변환 연산 예

(int) 3.8 + 5.7	8.7
3.8 + (int) 5.7	8.8
(int) 3.8 + (int) 5.7	8
(int) (3.8 + 5.7)	9
7 / 2	3
7.0 / 2	3.5
7 / (double) 2	3.5
(double) (7/2)	3.0

- 연산자 sizeof는 연산값 또는 자료형의 저장장소의 크기를 구하는 연산자이다. 연산자 sizeof의 결과값은 바이트 단위의 정수이다. 연산자 sizeof는 피연산자가 int와 같은 자료형인 경우 반드시 괄호를 사용해야 한다.

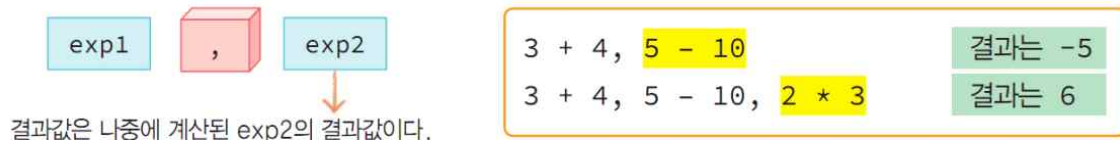
- 연산자 sizeof의 예시



sizeof (int)	결과는 4
sizeof (3.14)	결과는 8
sizeof a	결과는 2 (short a;)



- 콤마연산자 ,는 왼쪽과 오른쪽 연산식을 각각 순차적으로 계산하며 결과값은 가장 오른쪽에서 수행한 연산의 결과이다.
- 콤마연산자의 예시



- 복잡한 표현식의 계산에는 세가지 규칙이 있다.
  1. 괄호 우선 규칙
  2. 연산자 우선순위 규칙
  3. 연산자 결합성
- C언어의 연산자 우선순위 표

우선 순위	연산자	설명	분류	결합성(계산방향)
1	( ) [ ] . -> a++ a--	함수 호출 및 우선 지정 인덱스 필드(유니온) 멤버 지정 필드(유니온) 포인터 멤버 지정 후위 증가, 후위 감소	단항	-> (좌에서 우로)
2	++a --a ! ~ sizeof - + & *	전위 증가, 전위 감소 논리 NOT, 비트 NOT(보수) 변수, 자료형, 상수의 바이트 단위 크기 음수 부호, 양수 부호 주소 간접, 역참조		<- (우에서 좌로)
3	(형변환)	형변환		
4	* / %	곱하기 나누기 나머지	산술	-> (좌에서 우로)
5	+ -	더하기 빼기		-> (좌에서 우로)
6	<< >>	비트 이동	이동	-> (좌에서 우로)
7	< > <= >=	대소 비교	관계	-> (좌에서 우로)
8	== !=	동등 비교		-> (좌에서 우로)
9	&	비트 AND 또는 논리 AND	비트	-> (좌에서 우로)
10	^	비트 XOR 또는 논리 XOR		-> (좌에서 우로)
11		비트 OR 또는 논리 OR		-> (좌에서 우로)
12	&&	논리 AND(단락 계산)	논리	-> (좌에서 우로)
13		논리 OR(단락 계산)		-> (좌에서 우로)
14	? :	조건	조건	<- (우에서 좌로)
15	= += -= *= /= %= <<= >>= &=  = ^=	대입	대입	<- (우에서 좌로)
16	,	кома	кома	-> (좌에서 우로)

## 8. 조건문

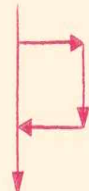
### (1) 제어문 개요

- 지금까지 배운 프로그램의 실행 순서의 원칙은 순차적 실행인데 이뿐만 아니라 선택과 반복 등 순차적인 실행을 변형하여 프로그램의 실행 순서를 제어하는게 제어문이다.
- 제어문의 종류는 조건선택과 반복, 분기처리로 나눌 수 있다.

#### 조건선택

조건에 대한 선택 구문


- if
- if else
- if else if
- nested if
- switch



#### 반복 순환

반복조건에 따라 일정영역의 반복 구문


- for
- while
- do while



#### 분기처리

지정된 영역으로 실행을 이동하는 구문

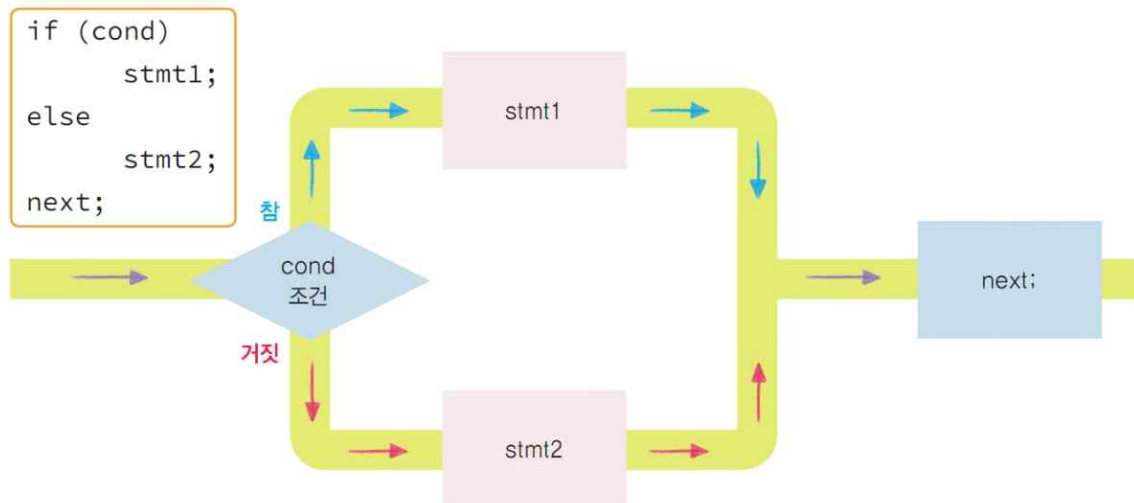
- break
- continue
- goto
- return



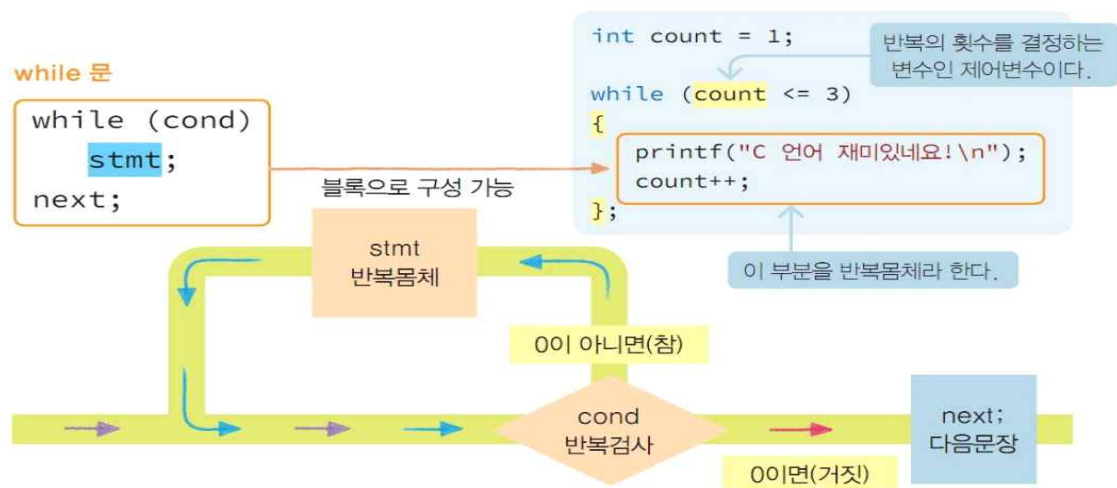
### (2) 조건에 따른 선택 if 문

- 문장 if는 조건에 따른 선택을 지원하는 구문이다. 조건식(cond)는 반드시 괄호가 필요하고, 참이면 실행되는 문장은 들여쓰기를 통해 작성해야 한다.
- if else는 1번 조건과 2번 조건중 선택하는 구문이다.

#### 조건문 if else



- 중첩된 if문이란 if문 내부에 if문이 존재하면 중첩된 if문이라 한다. 이런 경우 블록표시에 주의 해야한다. 여기서 else는 문법적으로 같은 블록 내에 다른 else가 없는 경우 가장 근접한 상위의 if문에 소속된 else로 해석한다.
- if문도 조건연산자의 역할을 할 수 있지만 간단한 경우에는 조건연산자를 사용하는게 더욱 편리하다.



### (3) 다양한 선택 switch 문

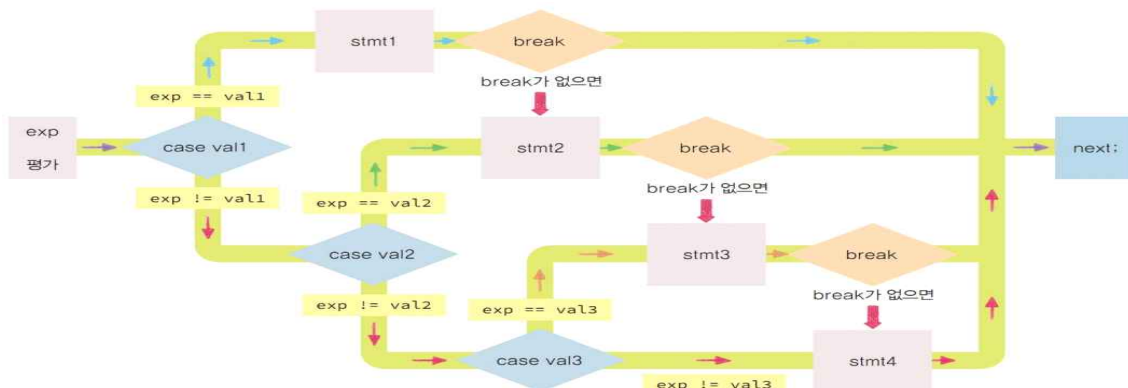
- switch문을 이용하면 문자 if else가 여러 번 계속 반복되는 구문을 좀 더 간략하게 구현이 가능하다. 주어진 연산식이 문자형 또는 정수형일 때 그값에 따라 case의 상수값과 일치하는 부분의 문장들을 수행하는 선택 구문이다.
- switch문의 흐름도

- switch 문에서는 break의 적절한 사용이 중요하다. case문 내부에 break문이 없다면 일치하는 case문을 실행하고, break문을 만나기 전까지 다음 case문을 실행하기 때문이다.

## 9. 반복문

### (1) 반복 개요와 while문

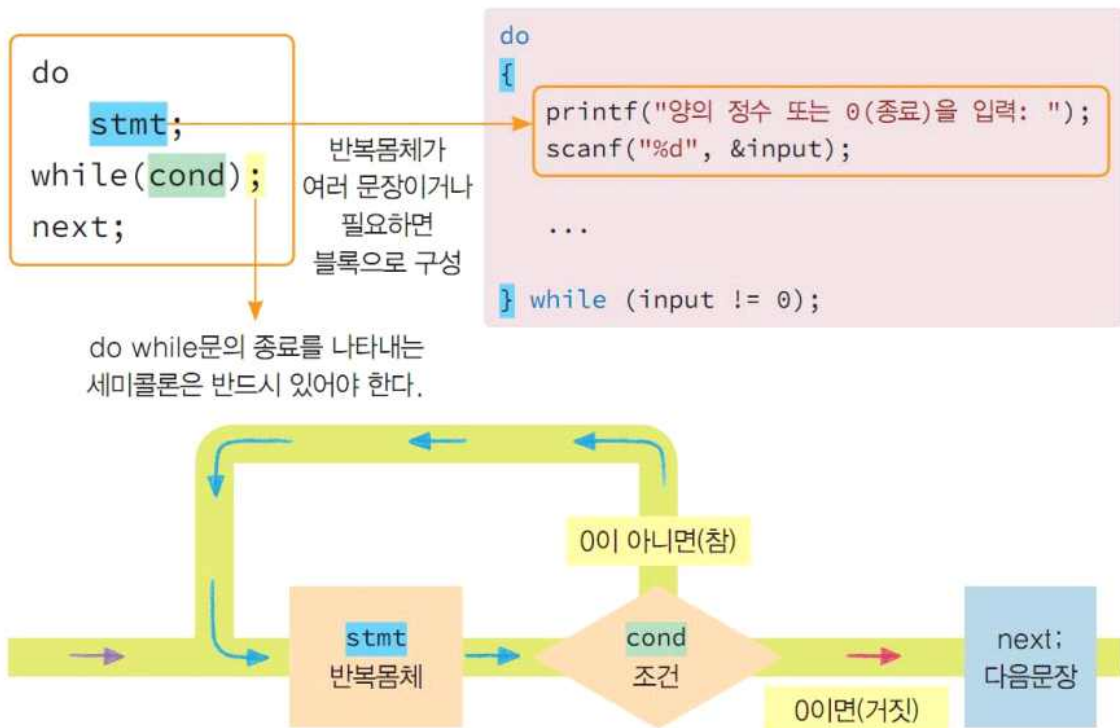
- 반복은 같거나 비슷한 일을 여러 번 수행하는 작업이다. C언어에서는 while, do while, for 세 가지의 반복문을 지원한다.
- while (cond) stmt; 는 반복조건인 cond를 평가하여 거짓이 될 때까지 반복한다.
- while문의 제어흐름



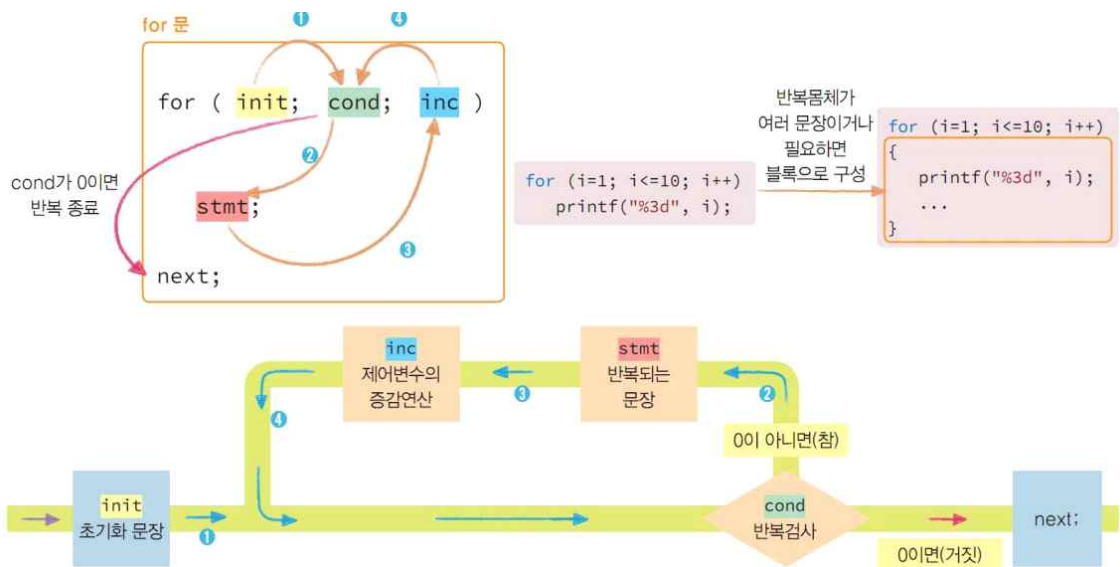


## (2) do while문과 for문

- do while문은 반복문체 수행 후에 반복조건을 검사한다. 그러므로 조건과 상관 없이 한 번은 실행한 후에 반복되는 것이다.
- do while문의 제어흐름



- for 문은 (init; cond; inc) stmt; 형식으로 주로 사용하는데 init에서는 초기화 cond는 조건 inc는 반복을 결정하는 변수의 증감을 수행하는데 사용된다.
- for 문의 제어흐름

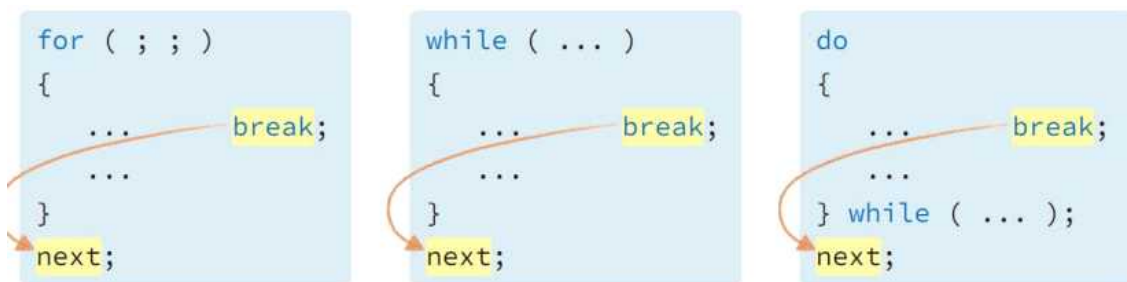


-for문과 while문은 서로 유사하지만 for문은 제어변수를 사용하고 초기화와 증감부분이 있는 반복문에 적합하고, while문은 반복횟수가 정해지지 않은채 특정한 조건에 따라 반복하는 구문에 적합하다.

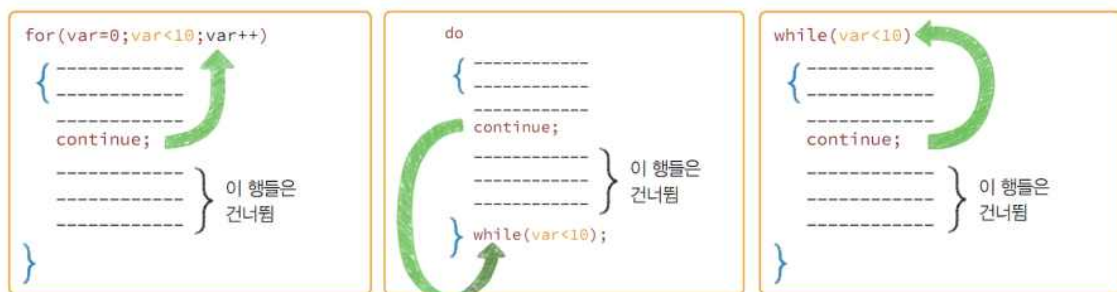
### (3) 분기문

- 분기문은 정해진 부분으로 바로 실행을 이동하는 기능을 수행한다. C가 지원하는 분기문은 break, continue, goto, return 문이 있다.

- break 문



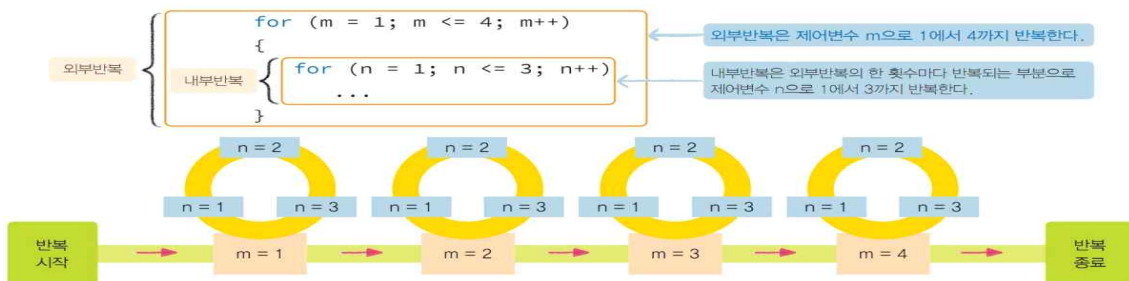
- continue 문



- 반복문의 내부에서 반복을 중단시키는 break 문, 반복문 내부에서 다음행들을 건너뛰고 다음반복으로가는 continue 문, 레이블이 위치한 다음 문장으로 실행순서를 이동하는 goto 문이 있다.

### (4) 중첩된 반복문

- 반복문 내부에 반복문이 또 있는 구문을 중첩된 반복문이라 한다.



## 10. 포인터 기초

### (1) 포인터 변수와 선언

- 포인터 변수는 주소값을 저장하는 변수로 일반 변수와 구별되어 선언방법이 다르다. 자료형과 포인터 변수 이름 사이에 연산자 \*를 삽입한다. 포인터 변수이며 간단히 포인터라고 부른다.

#### 포인터 변수선언

자료형 \*변수이름 ;

```
int *ptring;  
short *ptrshort;  
char *ptrchar;  
double *ptrdouble;
```

```
int *ptring; //가장 선호  
short*ptrshort;  
char * ptrchar;  
double *ptrdouble;
```

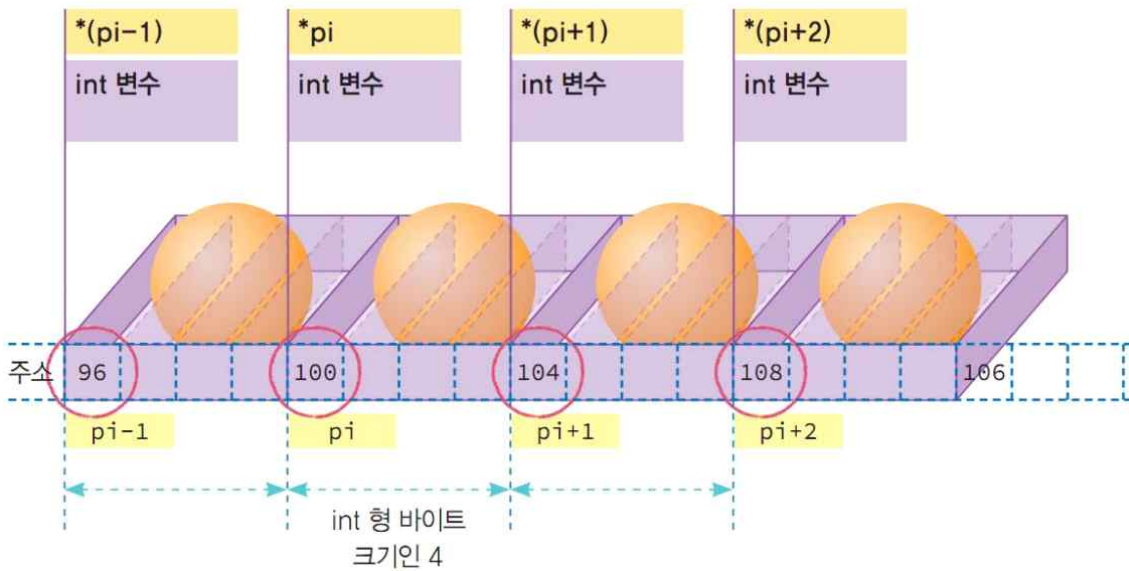
- 변수 자료형이 다르면 그 변수의 주소를 저장하는 포인터의 자료형도 달라야 한다.

### (2) 간접 연산자\*와 포인터 연산

- 여러 개의 포인터 변수를 한 번에 선언하기 위해서는 콤마 이후에 변수마다\*를 앞에 기술해야 한다. 포인터 변수도 다른 일반변수와 같이 지역변수로 선언하는 경우, 초기값을 대입하지 않으면 쓰레기값이 들어가므로 포인터 변수에 지정할 특별한 초기값이 없는 경우에 0번 주소값인 NULL로 초기값을 저장한다.
- 포인터 변수가 가리키고 있는 변수를 참조하려면 간접연산자 \*를 사용한다.
- 변수 data 자체를 사용해 자신을 참조하는 방식을 직접참조라 한다면 \*ptring를 사용해 변수data를 참조하는 방식을 간접참조라 한다.

```
int data = 100;  
int *ptring = &data;  
char *ptrchar = &ch;  
printf("간접참조 출력: %d %c\n", *ptring, *ptrchar);  
  
*ptring = 200;  
printf("직접참조 출력: %d %c\n", data, ch);
```

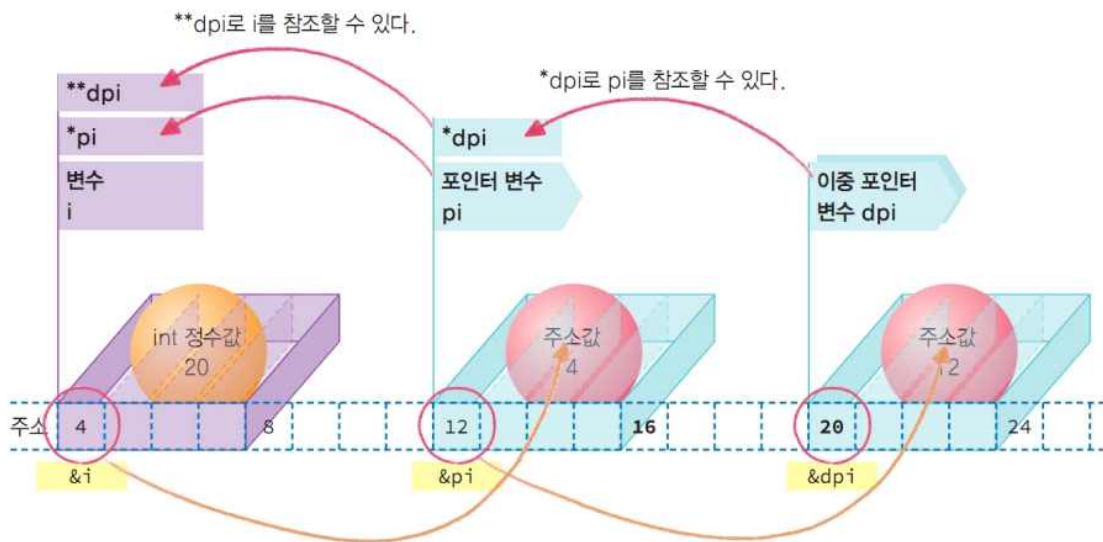
- 포인터 변수는 간단한 더하기와 뺄셈 연산으로 이웃한 변수의 주소 연산을 수행할 수 있다. 이 연산으로 이웃한 이전 또는 이후의 다른 변수를 참조한다.



### (3) 포인터 형변환과 다중 포인터

- 포인터 변수는 동일한 자료형끼리만 대입이 가능하기 때문에 자료형이 다른 경고가 발생한다. 자동으로 형변환이 불가능하며 필요하면 명시적으로 형변환을 수행한다.
- 포인터 변수의 주소값을 갖는 변수를 이중 포인터라 한다. 똑같이 이중포인터의 값을 갖는 변수를 삼중포인터라 하며 이러한 포인터의 포인터를 다중 포인터라 칭하고 \*의 여러 번 사용해 다중 포인터 변수를 선언한다.

```
int i = 20;
int *pi = &i;
int **dpi = &pi;
```



- 포인터 변수의 증감연산자와 간접연산자의 활용

연산식		결과값	연산 후 *p의 값	연산 후 p 증가
*p++	*(p++)	*p: p의 간접참조 값	변동 없음	p+1: p 다음 주소
++*p	*(++p)	*(p+1): p 다음 주소 (p+1) 간접참조 값	변동 없음	p+1: p 다음 주소
(*p)++		*p: p의 간접참조 값	*p가 1 증가	p: 없음
+++p	++(*p)	*p + 1: p의 간접참조 값에 1 증가	*p가 1 증가	p: 없음

- 키워드 const를 통해 일반변수와 같이 포인터 변수도 포인터 상수로 만드는데 가능하다.

```
int i = 10, j = 20;
```

```
❶ const int *pi = &i;
```

```
*pi = 20; //오류 발생
```

```
const int *pi
*pi가 상수로 *pi로 수정할 수 없음
오류: 식이 수정할 수 있는 lvalue여야 합니다.
```

```
❷ int const *pi = &i;
```

```
*pi = 20; //오류 발생
```

```
const int *pi
*pi가 상수로 *pi로 수정할 수 없음
오류: 식이 수정할 수 있는 lvalue여야 합니다.
```

```
❸ int* const pi = &i;
```

```
pi = &j; //오류 발생
```

```
int *const pi
오류: 식이 수정할 수 있는 lvalue여야 합니다.
```

## 11. 배열

### (1) 배열 선언과 초기화

- 배열은 여러 변수들이 같은 배열이름으로 일정한 크기의 연속된 메모리에 저장되는 구조이다.

- 배열을 이용하면 변수를 일일이 선언하는 번거로움을 해소할 수 있고, 배열을 구성하는 각각의 변수를 참조하는 방법도 간편하며 반복 구문으로 쉽게 참조 가능하다. 배열을 구성하는 항목을 원소라고 하고, 배열은 배열이름, 원소자료형, 배열크기로 이루어진다. 배열원소는 배열원소의 첨자번호로 쉽게 접근할 수 있다. -배열도 당연히 변수이므로 사용 전 선언을 반드시 해줘야 하고, 자료형 이름[크기]; 로 선언한다. 선언 시 초기값 지정이 없다면 양수로 명시해야 한다.



## - 배열 선언의 예시

**원소자료형**    배열이름[배열크기];

↑

배열크기는 리터럴 상수, 매크로 상수  
또는 이들의 연산식이 허용된다

```
#define SIZE 5

int score[10];
double point[20];
char ch[80];
float grade[SIZE];
int score[SIZE+1];
int degree[SIZE*2];
```

```
int n = 5;

int score[n];
double point[-3];
char ch[0];
float grade[3.2];
int score[n+2];
int degree[n*2];
```

오류발생

- 배열에서 첨자번호는 0부터 시작해 크기의 -1까지이다. 유효 범위를 벗어나 원소를 참조하게 되면 뭍법오류 없이 실행오류가 발생한다. 배열 선언 시에는 대괄호에 있는 정수가 배열의 크기이지만 선언 후 대괄호 안에는 첨자번호이다.
- 함수내부에서 배열 선언 후 원소에 초기값을 저장하지 않으면 쓰레기값이 저장되니 항상 초기값을 저장해 주어야 한다.
- C언어에서는 배열을 선언하면서 동시에 원소값을 손쉽게 저장하는 배열선언 초기화 방법을 제공한다.

### 배열선언 초기화

**원소자료형**    배열이름[배열크기] = {원소값1, 원소값2, 원소값3, 원소값4, 원소값5, ... } ;

↑

배열크기는 생략 가능하며, 생략 시 원소값의 수가 배열크기가 된다.

```
int grade[4] = {98, 88, 92, 95};
double output[] = {78.4, 90.2, 32.3, 44.6, 59.7, 98.9};
int cpoint[] = {99, 76, 84, 76, 68};
```

- 일반 배열선언과 달리 배열크기를 따로 지정하지 않아도 중괄호의 원소 수가 자동으로 배열의 크기가 된다.

## (2) 이차원과 삼차원 배열

- 이차원 배열은 테이블 형태의 구조를 나타낼 수 있으므로 행과 열의 구조로 표현한다.
- 이차원 배열의 선언은 2개의 대괄호가 필요하고 첫 번째 대괄호에는 배열의 행, 두 번째에는 배열의 열을 지정한다. 배열선언 시 초기값을 저장하지 않으면 반드시 행과 열의 크기는 명시되어야 한다. 행과 열이 따로 있으므로 이차원 배열을 참조하기 위해서는 두 개의 첨자가 필요하다.
- 이차원 배열 원소 참조 예시

외부반복 제어변수 i는 행을 순차적으로 참조

```
for (i = 0; i < ROWSIZE; i++)
{
    for (j = 0; j < COLSIZE; j++)
        printf("%d ", td[i][j]);
    puts("");
}
```

내부반복 제어변수 j는 한 행에서 열을 순차적으로 참조

- 이차원 배열을 선언하면서 초기값을 지정하는 방법을 중괄호를 중첩되게 이용하는 방법과 일차원 배열 같이 하나의 중괄호를 사용하는 방법이 있다.

```
int score[2][3] = {{30, 44, 67}, {87, 43, 56}};
```

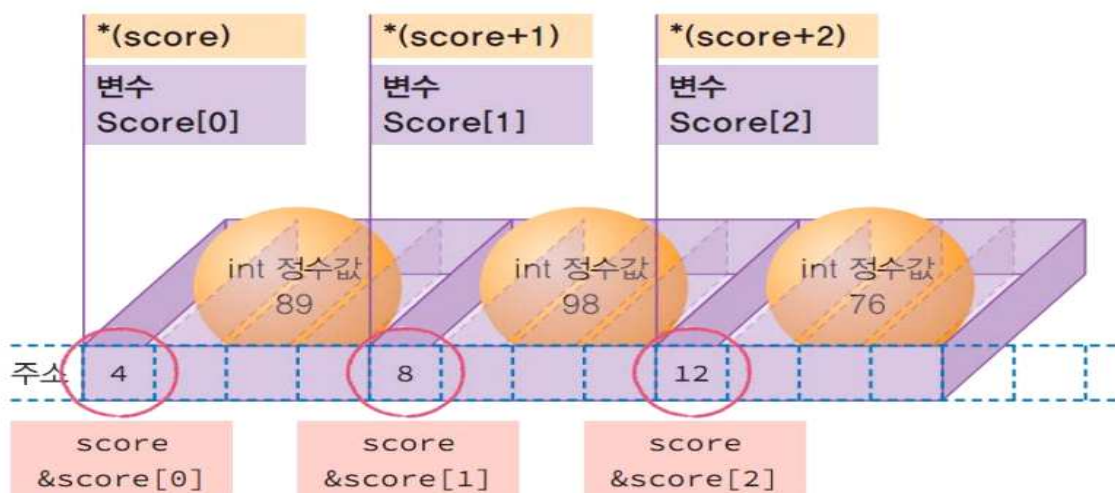


- C는 기본적으로 다차원 배열을 지원하는데 사차원 배열 이상을 사용하는 경우는 극히 드물다.
- 삼차원 배열은 선언과 초기화가 이차원 배열과 똑같지만 대괄호를 세 개를 사용해 배열을 만든다는 차이점이 있다.

### (3) 배열과 포인터 관계

- 배열은 실제로 포인터와 긴밀한 관계에 있고, 배열에서 배열이름 자체가 배열 첫 원소의 주소값인 상수이다.
- 배열 이름을 이용한 배열원소의 참조 예시

```
int score[] = {89, 98, 76};
```



score+1와 score+2의 차이는 원소크기인 4이다.

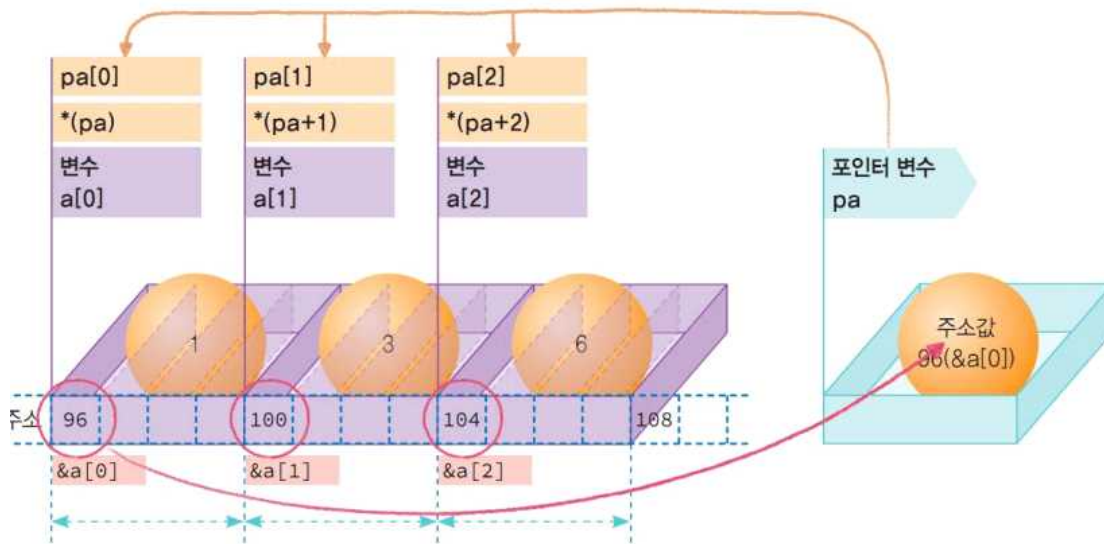
- 배열원소의 주소와 내용 값의 다양한 접근방법

배열 초기화 문장		int score[] = {89, 98, 76};		
배열 원소값		89	98	76
배열원소 접근방법	score[i]	score[0]	score[1]	score[2]
	*(score+i)	*score	*(score+1)	*(score+2)
주소값 접근 방법	&score[i]	&score[0]	&score[1]	&score[2]
	score+i	score	score+1	score+2
실제 주소값	base + 원소크기*i	만일 4라면	8 = 4+1*4	12 = 4+2*4

- 포인터 변수를 이용한 배열의 원소 참조

```
int a[4] = {1, 3, 6};
int *pa = &a[0];
```

```
printf("%d %d %d\n", *(pa), *(pa+1), *(pa+2)); //1, 3, 6 출력
printf("%d %d %d\n", pa[0], pa[1], pa[2]); //1, 3, 6 출력
```



#### (4) 포인터 배열

- 포인터 배열이란 주소값을 저장하는 포인터를 배열원소로 하는 배열이다.
- 포인터 배열 선언

```
int a = 5, b = 7, c = 9;

int *pa[3];

pa[0] = &a;    pa[1] = &b;    pa[2] = &c;
```



## 12. 함수 기초

### (1) 함수정의와 호출

- 이러한 특정한 작업을 처리하도록 작성한 프로그램 단위를 함수라 하고, 필요한 입력을 받아 원하는 어떤 기능을 수행한 후 결과를 반환하는 프로그램 단위.
- 함수는 라이브러리 함수와 사용자 정의 함수로 나뉘고 사용자가 직접 만든 함수를 사용하기 위해서는 함수선언, 함수호출, 함수정의가 필요하다.
- 역할을 잘 나눈 함수로 구성된 프로그램을 구조화된 프로그램이라 하며, 한번 정의된 함수는 여러 번 호출이 가능해 소스의 중복을 최소화 시킨다. 이러한 함수를 중심으로 프로그램을 짜는 것을 절차적 프로그래밍이라 한다.
- 함수 정의 예시

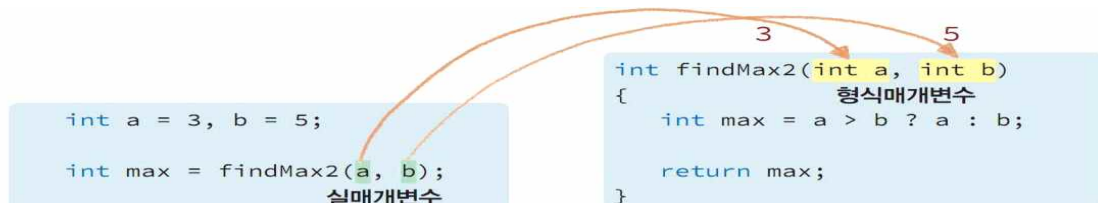
#### 함수정의



- 함수를 만들 때 반환값이 없다면 반환형으로 void를 기술한다. return 문장은 함수에서 반환값을 전달하는 목적과 함께 함수의 작업 종료를 알리는 문장이다.

### (2) 함수의 매개변수 활용

- 함수 매개변수는 함수를 호출하는 부분에서 함수몸체로 값을 전달하는 목적으로 사용된다. 필요한 경우 자료형과 변수명의 목록으로 나타내고 필요가 없다면 void를 기술한다.
- 값에 의한 호출의 함수에서 실매개변수의 값은 변하지 않는다.



함수호출 시 매개변수의 수와 자료형이 다르면 문법오류가 발생한다.

```
int max = findMax2();           //오류발생
int max = findMax2(2);         //오류발생
int max = findMax2(2.4);       //오류발생
int max = findMax2(2.4, 6.8);  //오류발생
```

- 함수의 매개변수로 배열을 전달한다면 한 번에 여러 개의 변수를 전달하는 효과가 있다.

#### 함수원형과 함수호출

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

...

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

... sum(data, 5);
```

#### 함수정의

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

    return total;
}
```

함수호출 시 배열이름으로 배열인자를 명시한다.

### (3) 재귀와 라이브러리 함수

- 함수구현에서 자신 함수를 호출하는 함수를 재귀함수라 한다. 재귀적 특성을 내포한 알고리즘에서 재귀 함수를 사용하면 문제를 더욱 쉽게 해결하는게 가능

$$n! \begin{cases} 0! = 1 \\ n! = n * (n-1)! \quad \text{for } (n \geq 1) \end{cases}$$

- 재귀함수는 함수의 호출이 계속되면 시간도 오래걸리고 메모리의 사용도 많다는 단점이 있다.

## 11. 문자와 문자열

### (1) 문자와 문자열

- C언어에서 char형 변수에 문자를 저장한다. 문자열을 저장하려면 문자의 모임인 문자 배열을 사용한다. 문자열을 선언하는 편리한 다른 방법을 살펴보면, 배열 선언 시 저장할 큰 따옴표를 사용해 문자열 상수를 바로 대입할 수 있다.

```
char c[] = "C language";
```

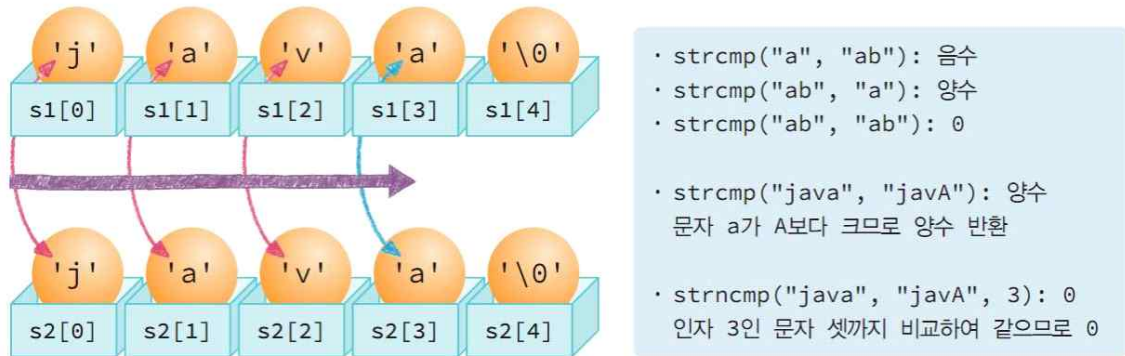
문자열: "C language"

문자열: "G"

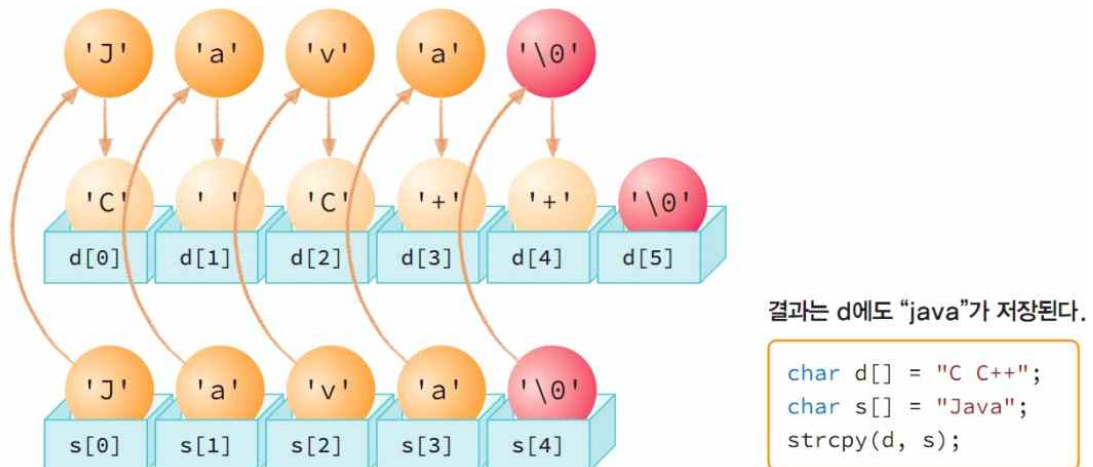
문자열: "Java"

## (2) 문자열 관련 함수

- 함수 strcmp()는 문자열 비교와 복사, 문자열 연결 등과 같은 다양한 문자열 처리는 헤더파일 string.h에 함수원형으로 선언된 라이브러리 함수로 제공



- 함수 strcpy()와 strncpy()는 문자열을 복사하는 함수이다. 함수 strcpy()는 앞 인자 문자열 dest에 뒤 인자 문자열 source를 복사한다.



- 함수 strcat()는 앞 문자열에 뒤 문자열의 null 문자까지 연결하여, 앞의 문자열 주소를 반환하는 함수이다.
- 함수 strtok()은 문자열에서 구분자인 문자를 여러 개 지정하여 토큰을 추출하는 함수이다.

### 문자열 분리 함수

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

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

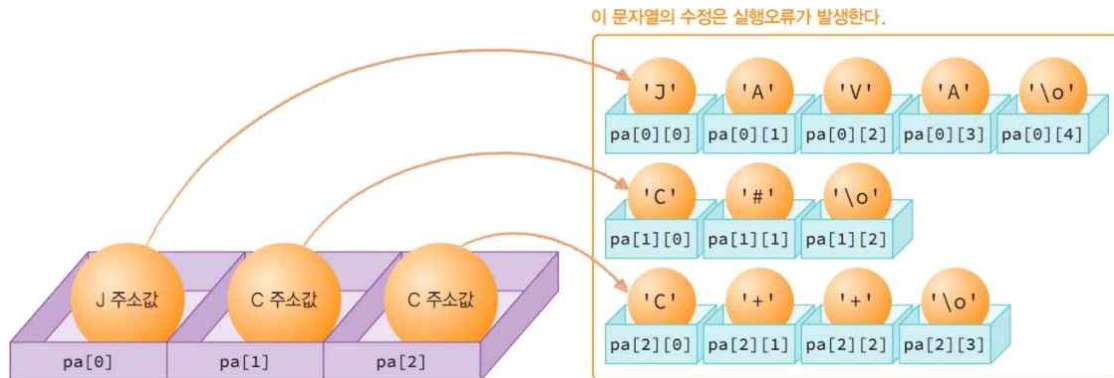
```
char * strtok_s(char * str, const char * delim, char ** context);
```

- 마지막 인자인 context는 함수 호출에 사용되는 위치 정보를 위한 인자이며, Visual C++에서는 앞으로 함수 strtok\_s()의 사용을 권장한다.

### (3) 여러 문자열 처리

- 여러 개의 문자열을 처리하는 하나의 방법은 문자 포인터 배열을 이용하는 방법이다. 다른 방법은 이차원 배열을 이용하는 것인데 예시로

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



## 12. 변수 유효범위

### (1) 전역변수와 지역변수

- 변수의 참조가 유효한 범위를 유효 범위라고 하고 크게 지역 유효 범위와 전역 유효 범위로 나눈다. 지역변수는 함수와 블록 내부에서 선언되는데 그 지역에서만 변수의 참조가 가능하다. 전역변수는 함수 외부에서 선언되는 변수이고, 한 프로젝트의 모든 함수나 블록에서 참조가 가능하다.
- 지역변수가 할당되는 메모리 영역을 스택이라 하고, 선언된 부분에서 자동생성 종료 후 자동삭제가 된다.
- 전역변수는 어디에서든 수정이 가능해 편하다는 장점이 있지만 사용자가 예상했던 값과 다른 값이 저장되면 해당 부분을 찾기가 힘들다는 단점이 있다.

### (2) 정적변수와 레지스터 변수

- 4가지의 기억부류 auto, register, static, extern에 따라 할당되는 메모리 영역과 메모리의 할당과 제거 시기가 결정된다.
- auto와 register는 지역변수에만 static은 지역과 전역 extern은 전역에만 사용이 가능하다.



- 키워드 register 변수는 일반 메모리가 아닌 CPU내부의 레지스터에 할당된다. 일반메모리를 사용할 수 없어서 주소연산자&를 사용할 수 없다.
- 변수 선언에서 키워드 static을 넣어 정적 변수를 선언 한다. 정적 변수는 프로그램 종료시 까지 메모리에서 제거되지 않고, 초기값을 지정하지 않으면 자동으로 자료형에 따라 0,W0,NULL값이 저장된다.

`static int svar = 1; //정적변수`

- 전역변수 선언 시 키워드 static을 가장 앞에 붙이면 정적 전역변수가 된다.
  - 정적 전역변수는 참조범위는 선언된 파일에만 한정되며 변수의 할당과 제거는 전역변수 특징을 갖는다.
- 지역변수 선언 시 키워드 static을 가장 앞에 붙이면 정적 지역변수가 된다.
  - 정적 지역변수는 참조범위는 지역변수이면서 변수의 할당과 제거는 전역변수 특징을 갖는다.
- 함수나 블록에서 정적으로 선언되는 변수가 정적 지역변수이다. 함수나 블록을 종료해도 메모리에서 제거되지 않고 계속 메모리에 유지 관리된다.
- 함수 외부에서 정적으로 선언되는 변수를 정적 전역변수라 하는데 선언된 파일 내부에서만 참조가 가능하다.

### (3) 메모리 영역과 변수 이용

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



#### -데이터영역

- 전역변수와정적변수가할당되는저장공간
- 메모리주소가낮은값에서높은값으로저장장소가할당
- 프로그램이시작되는시점에정해진크기대로고정된메모리영역이확보

#### -힙영역

- 동적할당dynamic allocation되는변수가할당되는저장공간
- 데이터영역과스택영역사이에위치



#### -스택영역

함수호출에 의한 형식매개변수 그리고 함수내부의 지역변수가 할당되는 저장공간  
힙영역과 스택영역은 프로그램이 실행되면서 영역크기가 계속적으로 변함  
메모리주소가 높은값에서 낮은값으로 저장장소가 할당  
함수호출과 종료에 따라 메모리가 할당되었다가 다시 제거되는 작업이 반복

#### -전역변수의 사용을 자제하고 지역변수를 주로이용

#### -레지스터변수

실행속도를 개선하고자하는 경우

#### -정적지역변수

함수나 블록내부에서 함수나 블록이 종료되더라도 계속적으로 값을 저장

#### -정적전역변수

해당파일 내부에서만 변수를 공유하는경우

#### -전역변수

프로그램의 모든영역에서 값을 공유하는경우

가능하면전역변수의사용을줄이는것이프로그램의이해를높일수있으며발생할 수 있는프로그램 문제를 줄일 수 있음

## 14. 구조체와 공용체

### (1) 구조체와 공용체

- 정수나 문자, 실수나 포인터 그리고 이들의 배열등을 묶어 하나의 자료형으로 이용하는 것이 구조체이다.
- 구조체(structure)를 자료형으로 사용하려면 먼저 구조체를 정의해야 한다.

#### 구조체 틀: 정의

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

구조체 정의 없이는 자료형 struct lecture를 사용할 수 없다.

#### 구조체를 자료형으로 사용

```
struct lecture datastructure;
```

- 구조체를 정의 했다면 이제 구조체 변수 선언이 가능하다.

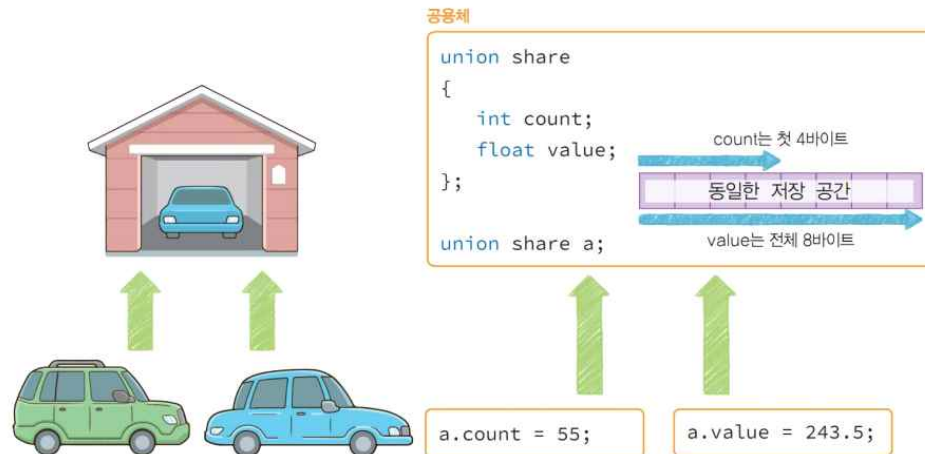
#### 구조체 자료형 변수 선언 및 초기화 구문

```
struct 구조체태그이름 변수명;
struct 구조체태그이름 변수명1, 변수명2, 변수명3, ...;

struct account yours;
struct account act1, act2, act3;
```

여러 변수의 선언도 가능하다.

- 공용체(union)란 동일한 저장 장소에 여러 자료형을 저장하는 방법으로 한번에 한종류만 저장하고 참조하는게 가능하다.

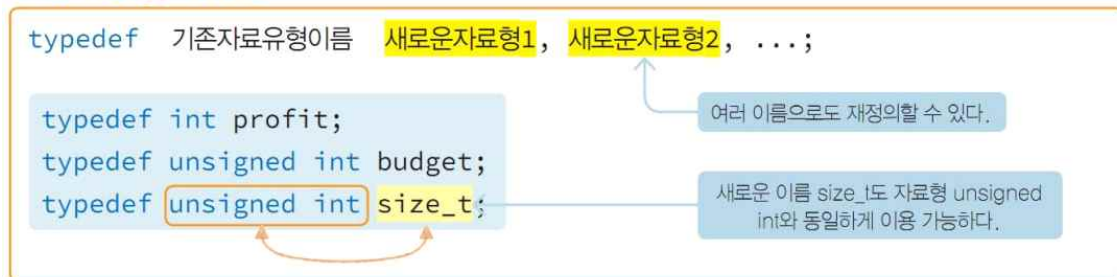


- 공용체의 크기는 멤버 중 가장 큰 자료형의 크기로 정해진다.

## (2) 자료형 재정의

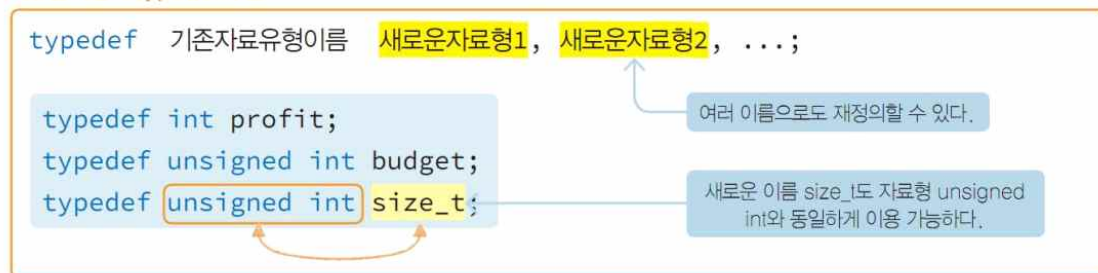
- 자료형 재정의 typedef는 이미 사용되는 자료 유형을 다른 새로운 자료형 이름으로 재정의할 수 있도록 하는 키워드이다.

### 자료형 재정의 typedef 구문



- 자료형을 재정의 하는 이유는 호환성과 편의성 증진을 위해 필요하기 때문
- 구조체 자료형 재정의

### 자료형 재정의 typedef 구문



## (3) 구조체와 공용체의 포인터와 배열

- 구조체 포인터는 구조체의 주소값을 저장할 수 있는 변수이다. 포인트 변수의 선언은 구조체와 일반 변수가 같다.
- 공용체 포인터 변수로 멤버를 접근하려면 접근연산자 ->를 사용한다.
- 구조체 배열은 동일한 구조체 변수가 여러개 필요할 때 선언하여 사용한다.

## 16. 함수와 포인터 활용

### (1) 함수의 인자전달 방식

- 함수 `increase(int origin, int increment)`

`origin += increment;` 를 수행하는 간단한 함수

함수호출시 변수 `amount`의 값 10이 매개변수인 `origin`에 복사되고,  
20이 매개변수인 `increment`에 복사

함수 `increase()` 내부 실행

매개변수인 `origin` 값이 30으로 증가

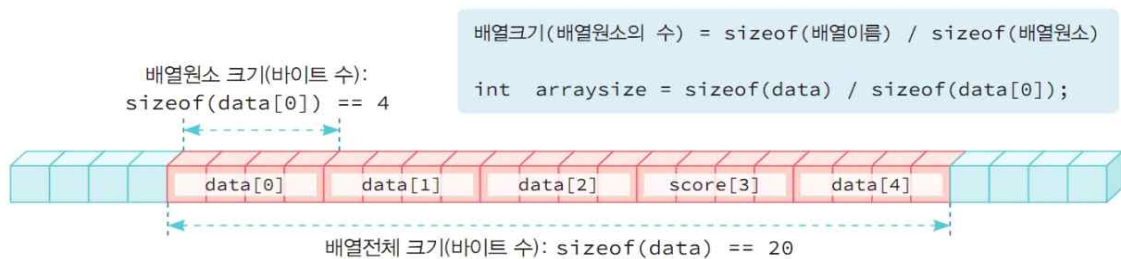
변수 `amount`와 매개변수 `origin`은 아무 관련성이 없음

`origin`은 증가해도 `amount`의 값은 변하지 않음

함수 외부의 변수를 함수 내부에서 수정할 수 없는 특징

- 배열 크기 계산법은 연산자 `sizeof`를 이용한 ( `sizeof`(배열이름) / `sizeof`(배열원소)) 결과는 배열 크기이다.

```
int data[] = {12, 23, 17, 32, 55};
```

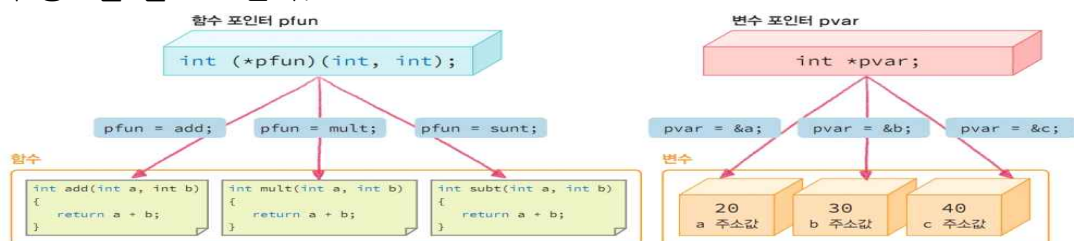


- 함수에서 인자의 수와 자료형이 결정되지 않은 함수 인자 방식을 가변인자라 하고, 구현하려면 선언, 처리 시작, 얻기, 처리 종료의 4단계를 거친다. 헤더파일 `<stdarg.h>`가 필요하다.

```
int vatest(int n, ...);
double vatum(char *type, int n, ...);
double vafun1(char *type, ..., int n); //오류, 마지막이 고정적일 수 없음
double vafun2(...); //오류, 처음부터 고정적일 수 없음
```

### (2) 함수 포인터와 void 포인터

- 함수 포인터는 함수의 주소값을 저장하는 포인터 변수다. 반환형과 인자목록의 정보를 필요로 한다.





-void 포인터는 자료형을 무시하고 주소값만을 다루는 포인터이다. 따라서 자료형과 상관없이 모든 자료형을 담을 수 있는 만능 포인터로 사용한다.

```
int m = 10;      double x = 3.98;
```

```
void *p = &m;
```

```
int n = *(int *)p; //int * 로 변환
```

```
n = *p; //오류
```

오류: "void" 형식의 값을 "int" 형식의 엔터티에 할당할 수 없습니다.

```
p = &x;
```

```
int y= *(double *)p; //double * 로 변환
```

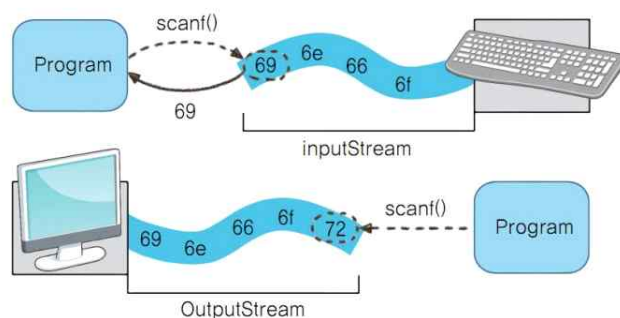
```
y = *p; //오류
```

오류: "void" 형식의 값을 "int" 형식의 엔터티에 할당할 수 없습니다.

## 17. 파일 처리

### (1) 파일 기초

- 파일은 텍스트 파일과 이진 파일 두 가지 유형으로 나뉜다.
- 이진 파일은 텍스트 파일과 다르게 그림, 동영상, 실행 파일이 각각의 목적에 맞는 자료가 이진 형태로 저장되는 파일이다.
- 자료의 입력과 출력은 자료의 이동이라고 볼 수 있는데 이 때 사용하는 통로가 입출력 스트림이라 한다.



- 프로그램에서 보조기억장치에 파일로 정보를 저장하거나 파일에서 정보를 참조하려면 파일에 대한 파일 스트림을 먼저 연결해야 한다. 파일스트림이란 보조기억장치의 파일과 프로그램을 연결하는 전송경로이다.
- 프로그램에서 특정한 파일과 파일 스트림을 연결하기 위해서는 함수 `fopen()` 또는 `fopen_s()`를 이용해야 한다. 파일 스트림 연결에 성공하면 정수 0을 반환하고 실패하면 양수를 반환한다.

```

struct _iobuf {
    char *_ptr;
    int _cnt;
    char *_base;
    int _flag;
    int _file;
    int _charbuf;
    int _bufsiz;
    char *_tmpfname;
};
typedef struct _iobuf FILE;

```

```

if ( (f = fopen(fname, "w")) == NULL )
{
    printf( "파일이 열리지 않습니다.\n" );
    exit(1);
};

```

- 함수 fclose()는 fopen()으로 연결한 파일스트림을 닫는다.

## (2) 텍스트 파일 입출력

- 함수 fprintf()와 fscanf() 또는 fscanf()\_s는 텍스트 파일에 자료를 쓰거나 읽기 위하여 사용한다.

### 함수 fprintf()와 fscanf() 함수원형

```

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

```

위 함수에서 \_File은 서식화된 입출력 스트림의 목적지인 파일이며, \_Format은 입출력 제어 문자열이며, 이후 기술되는 인자는 여러 개의 출력될 변수 또는 상수이다.

- 함수 fgets()는 한 행의 문자열을 입력받는 함수, fputs()는 한 행의 문자열을 출력하는 함수이다.

### 함수 fgets()와 fputs() 함수원형

```

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

```

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

```

char names[80];
FILE *f;

fgets(names, 80, f);
fputs(names, f);

```

- 함수 `feof()`은 EOF(End Of File) 표시를 검사하는 함수이다. 함수 `ferror()`는 파일 처리에서 오류가 발생했는지 검사하는 함수다

#### 함수 `feof()`와 `ferror()` 함수원형

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

- 함수 `feof()`은 `_File`의 EOF를 검사
- 함수 `ferror()`는 `_File`에서 오류발생 유무를 검사

```
while ( !feof(stdin) )  
{  
    ...  
    fgets(names, 80, stdin);    //표준입력  
}
```

- 함수 `fgetc()`는 파일로부터 문자 하나를 입력받고, `fputc()`는 문자하나를 파일로 출력하는 함수이다.

#### 함수 `fgetc()`와 `fputc()` 함수원형

```
int fgetc(FILE * _File);  
int fputc(int _Ch, FILE * _File);  
  
int getc(FILE * _File);  
int putc(int _Ch, FILE * _File);
```

- 함수 `fgetc()`와 `getc()`는 `_File`에서 문자 하나를 입력받는 함수
- 함수 `fputc()`와 `putc()`문자 `_Ch`를 파일 `_File`에 출력하는 함수