

# Portfolio

과목명	C 애플리케이션 구현
교수명	강환수 교수님
학과명	컴퓨터정보공학과
학 번	20193419
성 명	김범기

2019 학년도 1학기	전공	컴퓨터정보공학과	학부	컴퓨터공학부
과 목 명	C에플리케이션구현(2016003-PD)			
강의실 과 강의시간	월:6(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, 화 오전 11~12			
학과 교육목표	컴퓨터정보공학과는 국가 산업의 근간이 되는 정보기술 분야를 중심으로 컴퓨터 시스템 및 정보통신 분야의 체계적인 기초지식 및 이론 교육을 바탕으로 21세기 첨단 IT 분야에서 직무능력을 성공적으로 수행할 수 있는 실용적 기술과 직업능력을 갖춘 스마트앱개발 부문, 응용SW개발 부문의 창의 융합형 인력 육성을 목표로 한다.			
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, 이종규	인피니티북스	
수업시 사용도구	Visual C++			
평가방법	중간고사 30%, 기말고사 30%, 과제를 및 퀴즈 20% 출석 20%(학교 규정, 학업성적 처리 지침에 따름)			
수강안내	처음에는 지난 학기 과정의 복습부터 진행되겠지만, 기초적인 C 언어 학습이 먼저 되어 있으면 수강에 유리하다. 하지만 기초부터 수업이 진행되므로 선수 학습이 필수는 아닐.			

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

<b>7 주차</b>	<b>[7주]</b>
학습주제	합수
목표및 내용	10장 합수
미리읽어오기	교재 10장
과제,시험,기타	수업 중에 제시함
<b>8 주차</b>	<b>[중간고사 : 4/22 ~ 4/26]</b>
학습주제	중간고사
목표및 내용	중간고사
미리읽어오기	
과제,시험,기타	
<b>9 주차</b>	<b>[9주, 근로자의날(5/1) -&gt; 보강일(6/11)]</b>
학습주제	문자열
목표및 내용	11장 문자열
미리읽어오기	교재 11장
과제,시험,기타	수업 중에 제시함
<b>10 주차</b>	<b>[10주, 어린이날(5/5) -&gt; 보강일(6/10)]</b>
학습주제	변수 유효범위
목표및 내용	12장 변수 유효범위
미리읽어오기	교재 12장
과제,시험,기타	수업 중에 제시함
<b>11 주차</b>	<b>[11주]</b>
학습주제	구조체
목표및 내용	13장 구조체
미리읽어오기	교재 13장
과제,시험,기타	수업 중에 제시함
<b>12 주차</b>	<b>[12주, 개교기념일(5/20) -&gt; 보강일(6/12)]</b>
학습주제	합수와 포인터 활용
목표및 내용	14장 합수와 포인터활용
미리읽어오기	교재 14장
과제,시험,기타	수업 중에 제시함

<b>13 주차</b>	<b>[13주]</b>
학습주제	파일처리
목표및 내용	15장 파일처리
미리읽어오기	교재 15장
과제,시험,기타	수업 중에 제시함
<b>14 주차</b>	<b>[14주, 현충일(6/6) -&gt; 보강일(6/13)]</b>
학습주제	항상심화강좌(통적활당)
목표및 내용	16장 통적활당
미리읽어오기	교재 16장
과제,시험,기타	수업 중에 제시함
<b>15 주차</b>	<b>[기말고사 : 6/14 ~ 6/20]</b>
학습주제	기말고사
목표및 내용	기말고사
미리읽어오기	
과제,시험,기타	..
<b>수업지원 안내</b>	장애 학생을 위한 별도의 수강 지원을 받을 수 있습니다. 언어가 문제가 되는 학생은 글로 된 과제 안내, 확대문자 시험지 제공 등의 지원을 드립니다.

# C 애플리케이션구현 포트폴리오 목차

## Chapter 01. 프로그래밍 언어 개요 .....

- 1.1 프로그램이 무엇일까
- 1.2 언어의 계층과 번역
- 1.3 왜 C언어를 배워야 할까
- 1.4 프로그래밍 자료 표현
- 1.5 소프트웨어 개발
- 1.6 다양한 '프로그래밍 언어'

## Chapter 02. C 프로그래밍 첫걸음 .....

- 2.1 프로그램 구현 과정과 통합개발환경
- 2.2 비주얼 스튜디오 설치와 C프로그램의 첫 개발
- 2.3 C프로그램의 이해와 디버깅 과정

## Chapter 03. 자료형과 변수 .....

- 3.1 프로그래밍 기초
- 3.2 자료형과 변수선언
- 3.3 기본 자료형
- 3.4 상수 표현방법

## Chapter 04. 전처리와 입출력 .....

- 4.1 전처리
- 4.2 출력함수 printf()
- 4.3 입력 함수 scanf()

## Chapter 05. 연산자 .....

- 5.1 연산식과 다양한 연산자
- 5.2 관계와 논리, 조건과 비트연산자
- 5.3 형변환 연산자와 연산자 우선순위

## Chapter 06. 조건 .....

- 6.1 제어문 개요
- 6.2 조건에 따른 선택 if문
- 6.3 다양한 선택 switch문

## Chapter 07. 반복 .....

- 7.1 반복 개요와 while 문
- 7.2 do while문과 for 문
- 7.3 분기문
- 7.4 중첩된 반복문

## Chapter 08. 포인터 기초 .....

- 8.1 포인터 변수와 선언
- 8.2 간접 연산자 \*와 포인터 연산
- 8.3 포인터 형변환과 다중 포인터

## Chapter 09. 배열 .....

- 9.1 배열 선언과 초기화
- 9.2 이차원과 삼차원 배열
- 9.3 배열과 포인터 관계
- 9.4 포인터 배열과 배열 포인터

## Chapter 10. 함수 기초 .....

- 10.1 함수정의와 호출
- 10.2 함수의 매개변수 활용
- 10.3 재귀와 라이브러리 함수

## Chapter 11. 문자와 문자열 .....

- 11.1 문자와 문자열
- 11.2 문자열 관련 함수
- 11.3 여러 문자열 처리

## Chapter 12. 변수 유효범위 .....

- 12.1 전역변수와 지역변수
- 12.2 정적 변수와 레지스터 변수
- 12.3 메모리 영역과 변수 이용

## Chapter 13. 구조체와 공용체 .....

- 13.1 구조체와 공용체
- 13.2 자료형 재정의
- 13.3 구조체와 공용체의 포인터와 배열







## 1.2 언어의 계층과 번역 .....

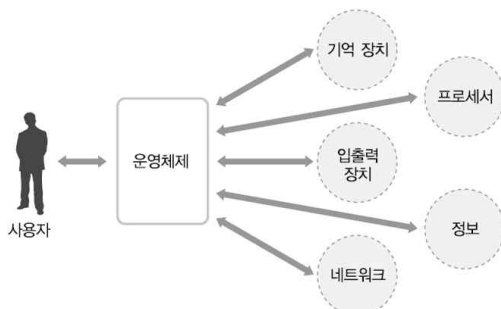
컴퓨터는 고철 덩어리인 하드웨어와 이 하드웨어를 작동하게 하는 소프트웨어로 구성된다. 이해를 돕기 위해 예를 들면 사람의 경우 소프트웨어는 영혼, 하드웨어는 육체로 비유할 수가 있는데 영혼없는 육체는 아무 의미가 없듯이 소프트웨어 없는 하드웨어인 컴퓨터는 아무것도 할 수 없는 단순한 고철 덩어리에 불과하다.



하드웨어의 중요한 구성 요소로는 중앙처리장치(CPU), 주기억장치(main memory), 보조기억장치(secondary memory), 입력장치(input device), 출력장치(output device)를 들 수 있다. 중앙처리장치는 연산을 수행하는 연산장치, 연산을 제어하는 제어장치로 구성되며 이 중앙처리장치의 칩을 프로세서라고 한다. 또한 소프트웨어는 컴퓨터가 수행할 작업을 지시하는 전자적 명령어들의 집합으로 구성된 프로그램을 말한다.

소프트웨어는 컴퓨터가 특정 작업을 수행할 수 있도록 해주는 전자적인 명령어 집합으로 구성되며 컴퓨터의 하드웨어가 해야 할 작업 내용을 지시한다. 소프트웨어는 크게 응용소프트웨어와 시스템 소프트웨어로 나뉘며 응용소프트웨어란 문서 작성이나 인터넷 검색, 게임하기 동영상 보기 등과 같은 특정 업무에 활용되는 소프트웨어이고 시스템소프트웨어란 컴퓨터가 잘 작동하도록 도와주는 기본 소프트웨어를 말한다.

운영체제란 컴퓨터 하드웨어 장치의 전반적인 작동을 제어하고 조정하며, 사용자가 최대한 컴퓨터를 효율적으로 사용할 수 있도록 돕는 시스템프로그램이다. 하드웨어와 응용프로그램간의 인터페이스 역할을 하면서 CPU, 주기억장치, 입출력장치 등의 컴퓨터 자원과 함께 다양한 프로그램과 네트워크 등을 관리한다.

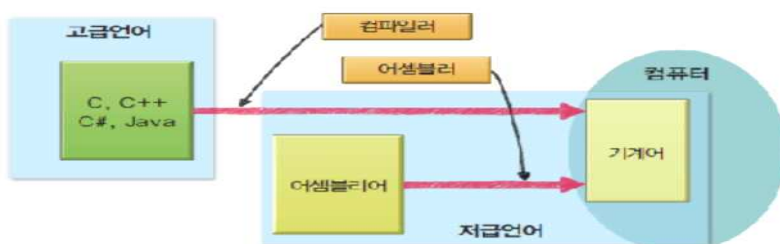


[그림 1-1] 운영체제의 역할



컴퓨터는 전기의 흐름을 표현하는 1과 흐르지 않음을 의미하는 0으로 표현되는 기계어만 인식할 수 있으며 프로그래밍 언어를 사용하는 프로그래머와 기계어를 사용하는 컴퓨터가 의사교환하기 위해서는 통역사인 컴파일러가 필요하다. 어셈블리어는 기계어를 좀 더 사람이 이해하기 쉬운 기호형태로 일대일 대응시킨 프로그래밍 언어이며 기계어 보다는 프로그래밍이 훨씬 용이하다.

저급언어란 컴퓨터의 CPU에 적합하게 만든 기계어와 어셈블리 언어를 저급언어라고 하며 고급언어는 CPU에 의존하지 않고 우리 사람이 보다 쉽게 이해할 수 있도록 만들어진 언어이다.



컴파일러는 고급언어로 작성된 프로그램을 기계어 또는 목적코드로 바꾸어 주는 프로그램이며 어셈블러는 어셈블리어로 작성된 프로그램을 기계어로 바꾸어 주는 프로그램이다.

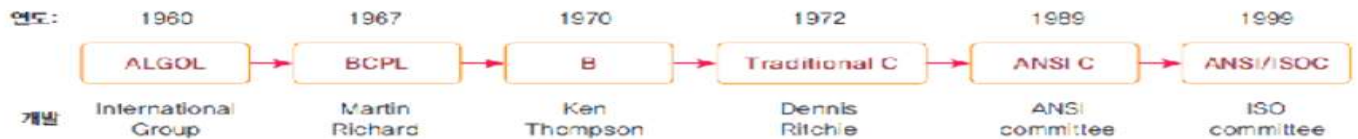
### 1.3 왜 C언어를 배워야 할까? .....



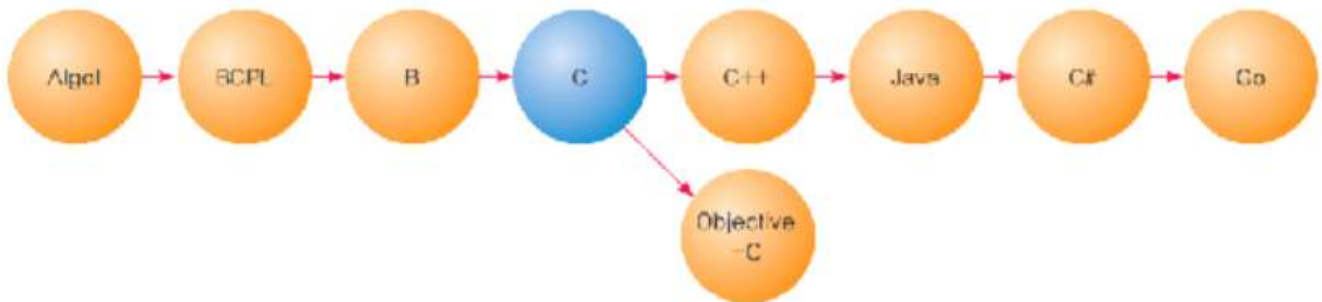
1972년 미국전신전화국(AT&T)의 벨 연구소(Bell Lab)에 근무하던 데니스 리치는 시스템 PDP-11에서 운용되는 운영체제인 유닉스(Unix)를 개발하기 위해 C 언어를 개발하였다.

“어셈블리 언어 정도의 속도를 내며, 좀 더쉽고, 서로 다른 CPU에서도 작동되는 프로그래밍 언어가 필요하다.”

C언어는 켄 톰슨이 1970년 개발한 B 언어에서 유래하였다. B언어 1970년 BCPL 프로그래밍 언어에 기반을 두고 개발된 언어 BCPL(Basic Combined Programming Language)은 캠브릿지 대학의 마틴 리차드(Martin Richards)가 1969년 개발한 프로그래밍 언어로 CPL(Combined Programming Language) 언어에서 발전하였으며 1960년 초에 개발된 CPL은 1960년에 개발된 Algol 60으로부터 많은 영향을 받은 언어이다.



1983년 얀 스트로스트룹(Bjarne Stroustrup)은 C 언어에 객체지향프로그래밍 개념을 확장시킨 프로그래밍 언어 C++를 개발하였으며 1995년에는 선 마이크로시스템즈(Sun Microsystems) 사는 C++ 언어를 발전시켜 인터넷에 적합한 언어인 자바Java)를 발표하였다.

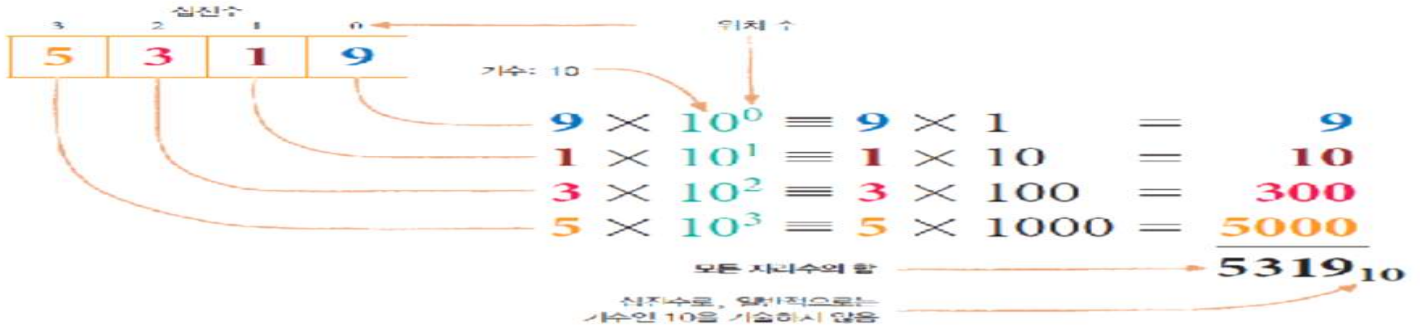


C언어는 함수 중심으로 구현되는 절차지향 언어(procedural language)이며 시간의 흐름에 따라 정해진 절차를 실행한다는 의미문제의 해결 순서와 절차의 표현과 해결이 쉽도록 설계된 프로그램 언어라는 특징을 갖고 있는 절차지향 언어이다. 또한 크기도 작으며, 메모리도 적게 효율적으로 사용하여 실행하며 속도가 빠르다는 장점이 있어 간편하고 효율적인 언어라고도 할 수 있다. C로 작성된 소스는 별다른 수정 없이 다양한 운영체제의 여러 플랫폼에서 제공되는 컴파일러로 번역(compile)해 실행할 수 있으며 다양한 CPU와 플랫폼의 컴파일러를 지원한다. 하지만 다소 어렵다는 단점이 있다.

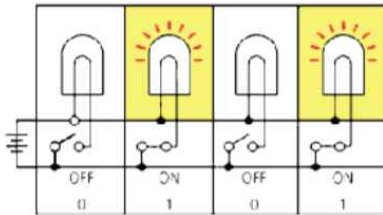
우리가 C언어를 배워야 하는 이유는 많이 쓰이는 자바나 C#,Objective-C 등이 그 뿌리는 C이다. C 언어를 알면 자바나 C#, Objective-C 뿐만 아니라 그 이후의 프로그래밍 언어들은 가지를 뻗어나가듯 습득이 매우 쉬워질 수 있으며 임베디드 시스템(embedded system)에서부터 응용프로그램(application program), 운영체제와 같은 시스템 소프트웨어(system software) 개발 등 여러 분야에 널리 사용되고 있다. 또 단순히 C 언어의 문법뿐 아니라 일반적인 프로그래밍 기초 지식을 학습한다고 생각할 수 있으며 C 언어의 학습은 여러분이 프로그래머나 정보기술 개발자가 되기 위한 초석을 다지는 기회라고 할 수 있다.

## 1.4 프로그래밍의 자료표현 .....

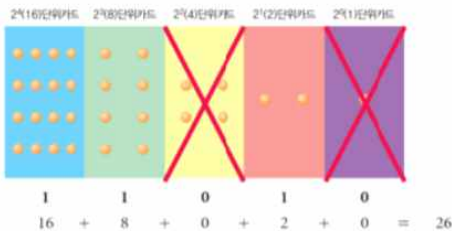
십진수는 수에서 하나의 자릿수(digits)에 사용하는 숫자가 0, 1, 2, 3, 4, 5, 6, 7, 8, 9까지 열 개이므로 십진수라고 부른다. 십진수 5319는 다음 그림으로 설명하듯이,  $1000(10^3)$ 인 것이 5개,  $100(10^2)$ 인 것이 3개,  $10(10^1)$ 인 것이 1개, 마지막으로  $1(10^0)$ 인 것이 9개 모인 수를 말한다.



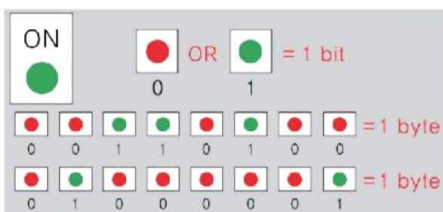
프로그래밍에서는 일상생활같이 십진수를 사용할 수 있지만, 시스템 내부에서는 십진수가 아닌 이진수를 사용하여 저장한다. 일상생활에서 '참과 거짓', '남자와 여자', '스위치의 on 과 off'와 같이 두 가지로 표현되는 것이 있다. 컴퓨터는 전기적 소자인 트랜지스터로 자료값을 저장하므로 전기가 흐르거나 흐르지 않는 두 가지 전기 신호만으로 자료를 처리하고 저장한다. 이와 같이 디지털 신호에서 전기가 흐를 경우 '참'을 의미하는 '1', 흐르지 않을 경우 '거짓'을 의미하는 '0'으로 표현되므로, 컴퓨터 내부에서 처리하는 숫자 0과 1을 표현하는 이진수 체계를 사용한다.



컴퓨터는 논리의 조합이 간단하고 내부에 사용되는 소자의 특성상 편리하기 때문에 이진법을 사용하는 것이 가장 합리적이고 효율적인 방식이다.



이진수는 수의 자릿수에 사용할 수 있는 숫자가 0과 1, 2개 이므로 이진수라 한다. 이진수에서 가장 오른쪽의 단위는 ( $2^0$ )단위이며 왼쪽으로 갈수록 2의 제곱으로 2, 4, 6, 8, 16 과 같이 바로 오른쪽의 2배로 증가하는 자릿수이다. 하지만 이진수는 길이가 길다는 문제가 있다.



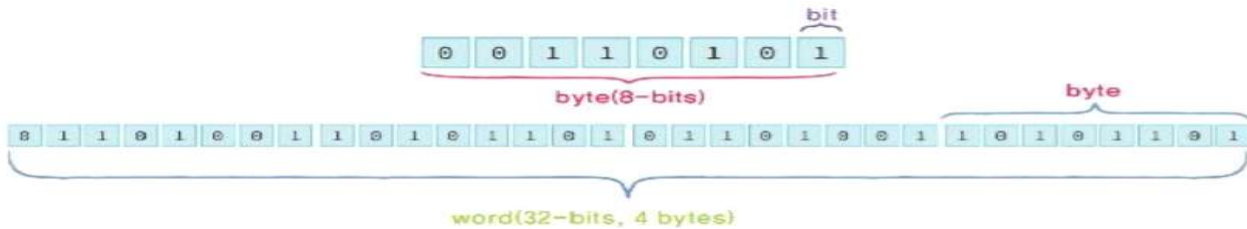
비트(bit)는 Binary digit의 합성어이며 가장 작은 기본 정보 단위이다. 즉 전기의 흐름 상태인 온(on)과 오프(off)를 표현하는 단위가 비트로 1과 0인 이진수로 표현이 가능하다.

단위	약자	표현	바이트(byte)	근접
1 bit	1 b (lower case)	0 또는 1		
1 Nibble		4 bits	½ byte	
1 Byte	1B (upper case)	8 bits 또는 2 Nibbles 또는 $2^3$ bits	1 byte	
1 Kilobyte	1 KB	$2^{10}$ bytes	1,024 byte	1 thousand bytes
1 Megabyte	1 MB	$2^{20}$ bytes	1,048,576 byte	1 million bytes
1 Gigabyte	1 GB	$2^{30}$ bytes	1,073,741,824 byte	1 trillion bytes
1 Terabyte	1 TB	$2^{40}$ bytes	1,099,511,627,776 byte	1 quadrillion bytes
1 Petabyte	1 PB	$2^{50}$ bytes	1,125,899,906,842,624 byte	1 quintillion bytes
1 Exabyte	1 EB	$2^{60}$ bytes	1,152,921,504,606,846,976 byte	1 sextillion bytes
1 Zettabyte	1 ZB	$2^{70}$ bytes	1,180,591,620,717,411,303,424 bytes	1 septillion bytes

바이트는 정보용량의 단위이며 킬로, 메가, 기가, 테라 등은 그 크기를 표현한다.

이 그림은 저장용량의 단위이다.

8개의 비트가 모인 정보 단위를 바이트(byte)라고 하며 비트는 총  $2^8=256$ 가지의 정보 종류를 저장한다. 일반적으로 바이트가 4개, 8개 모이면 워드(word)라고 하지만 시스템마다 그 크기는 다를 수 있음을 기억해야 한다.



참과 거짓을 의미하는 두 가지 정보를 논리값이라고 하며 0과 1을 각각 거짓과 참으로 표현 가능하다. 부울대수는 컴퓨터가 정보를 처리하는 방식에 대하여 이론적인 배경을 제공하며, 0과 1 두 값중 하나로 한정된 변수들의 상관 관계를 AND,OR,NOT 등의 여러 연산자를 이용하여 논리적으로 나타낸다. AND 연산은 두 개의 항이 모두 '1' 이어야 '1'이며, OR 연산은 둘중하나만 1이면 결과가 1이다. 항이 하나인 NOT 연산 같은 경우에는 항이 0이면 1으로 1이면 0으로 결과를 반환하는 연산이다.

<table><tr><th>A</th><th>B</th><th>A AND B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table> <p>AND</p>	A	B	A AND B	0	0	0	0	1	0	1	0	0	1	1	1	<table><tr><th>A</th><th>B</th><th>A OR B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table> <p>OR</p>	A	B	A OR B	0	0	0	0	1	1	1	0	1	1	1	1	<table><tr><th>A</th><th>NOT B</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table> <p>NOT</p>	A	NOT B	0	1	1	0	<table><tr><th>A</th><th>B</th><th>A NAND B</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table> <p>NAND</p>	A	B	A NAND B	0	0	1	0	1	1	1	0	1	1	1	0	<table><tr><th>A</th><th>B</th><th>A NOR B</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table> <p>NOR</p>	A	B	A NOR B	0	0	1	0	1	0	1	0	0	1	1	0	<table><tr><th>A</th><th>B</th><th>A XOR B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table> <p>XOR</p>	A	B	A XOR B	0	0	0	0	1	1	1	0	1	1	1	0
A	B	A AND B																																																																																				
0	0	0																																																																																				
0	1	0																																																																																				
1	0	0																																																																																				
1	1	1																																																																																				
A	B	A OR B																																																																																				
0	0	0																																																																																				
0	1	1																																																																																				
1	0	1																																																																																				
1	1	1																																																																																				
A	NOT B																																																																																					
0	1																																																																																					
1	0																																																																																					
A	B	A NAND B																																																																																				
0	0	1																																																																																				
0	1	1																																																																																				
1	0	1																																																																																				
1	1	0																																																																																				
A	B	A NOR B																																																																																				
0	0	1																																																																																				
0	1	0																																																																																				
1	0	0																																																																																				
1	1	0																																																																																				
A	B	A XOR B																																																																																				
0	0	0																																																																																				
0	1	1																																																																																				
1	0	1																																																																																				
1	1	0																																																																																				

Gate	Symbol	Operator
and		$A \cdot B$
or		$A + B$
not		$\bar{A}$

Gate	Symbol	Operator
nand		$\overline{A \cdot B}$
nor		$\overline{A + B}$
xor		$A \oplus B$

논리 연산은 AND 게이트, OR 게이트, NOT 게이트와 같은 논리 회로로 그릴 수 있으며 NOT AND를 의미하는 NAND 게이트, NOT OR를 의미하는 NOR 게이트 EXCLUSIVE OR를 의미하는 XOR 게이트 등도 조합하여 디지털 회로를 구성할 수 있다.

문자 인코딩 방식이란 문자를 하나의 숫자인 코드로 지정하여 처리하는 방식이며 대표적으로 두 개의 인코딩 방식이 있는데 먼저 아스키(ASCII: American Standard Code for Information Interchange) 코드는 1967년에 표준으로 제정하여 1986년에 마지막으로 개정하였으며 아스키는 초창기에 7비트 인코딩(8비트중 7비트만 데이터 비트로 사용)방식 즉 33개의 출력 불가능한 제어문자들과 공백을 비롯한 95개의 출력 가능한 문자들로 이루어져 총 128개의 코드로 구성되어 있었지만 아스키코드값이 주로 그래픽에 관련된 문자와 선 그리기에 관련된 문자가 추가되어 256개로 확장하여 8비트 인코딩을 사용할 수 있게끔 확장되었다. 두 번째로 유니코드는 전 세계의 모든 문자를 컴퓨터에서 일관되게 표현하고 다룰 수 있도록 설계된 산업 표준이다. 유니코드 협회(Unicode Consortium)가 제정하여 1991년 버전 1.0이 발표되었으며 동양권의 컴퓨터 관련 시장을 쉽게 접근하기 위해서도 미국 등의 유수의 S/W, H/W업체에게 문자 코드 문제는 가장 시급하게 해결되어야 할 걸림돌이다. 전 세계의 문자를 모두 표현하기 위해 2바이트 즉, 16비트로 확장된 코드 체계가 유니코드 1996년 65,536자의 코드영역을 언어학적으로 분류하였고 한글 완성자 11,172자의 한글과 중국, 일본을 포함해 세계 유수의 언어 문자를 배열해 만든 유니코드는 국제표준화기구(ISO: International Organization for Standardization)에 상정 확정되었으며 현재 계속 수정, 보완 중이라고 한다.

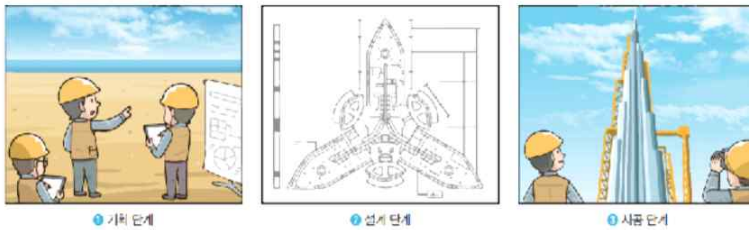


## 1.5 소프트웨어 개발 .....

소프트웨어는 보통 '프로그램'이라고 부르는 것 외에도 데이터와 문서를 포함하는 포괄적인 개념이다. 소프트웨어의 교육이 모두 프로그래머를 양성하기 위한 교육만은 아니며 프로그래밍 언어 습득 자체보다 사고를 절차화 하는 과정을 통해 논리력과 문제를 해결하는 능력 등 디지털 시대에 필요한 창의적 사고를 할 수 있는 기본적 소양을 증진 시킨다.



알고리즘(algorithm)이란 어떠한 문제를 해결하기 위한 절차나 방법으로 명확히 정의된 유한개의 규칙과 절차의 모임이다. 컴퓨터 프로그램은 특정한 업무를 수행하기 위한 정교한 알고리즘의 집합이라고 간주할 수 있다.



소프트웨어 공학이란 공학적 원리에 의하여 소프트웨어를 개발하는 학문이며 소프트웨어 개발과정인 분석, 설계, 개발, 검증, 유지보수 등 개발수명주기 전반에 걸친 계획·개발·검사·보수·관리, 방법론 등을 연구하는 분야이다.



첫 번째인 요구사항 분석에서는 시스템을 사용할 사용자의 요구사항을 파악하여 분석하는 단계이다. 두 번째인 설계단계에서 프로그래머는 알고리즘을 이용하여 소프트웨어를 설계한다. 구현 단계는 흐름도 또는 의사코드를 컴퓨터가 이해할 수 있는 자바나 C와 같은 특정한 프로그래밍 언어로 개발하는 단계이다. 검증 단계에서는 프로그램의 소프트웨어 요구사항에 얼마나 부합하는지, 프로그램이 안정적으로 작동하는지를 검사하는 단계이다. 마지막은 프로그램의 문서화 및 유지보수 단계이다.

기호	기능	기호	기능
	터미널 순서도의 시작과 끝을 표시		처리 가중 연산, 데이터 이동 등을 처리
	판단 여러 가지 경로 중 하나 의 경로 선택을 표시		입·출력 데이터의 입력 및 출력 표시
	흐름선 처리간의 연결 기능을 표시		연결자 흐름이 다른 곳과 연결되 는 입출구를 나타냄
	서류 시류를 머쳐모라는 입출 력 표시		준비 가역장소, 초기값 등 작 업의 준비 과정을 나타냄
	수용일벽 콘솔에 의한 입력		전송카드 전송카드의 입출력

문제를 해결하기 위한 절차나 방법의 모임인 알고리즘은 우리가 사용하는 자연어 또는 흐름도나 의사코드 등을 사용하여 표현될 수 있다.

의사코드는 슈도코드라고도 하며 특정 프로그래밍 언어의 문법을 따르지 않고 간결한 특정 언어로 코드를 흉내 내어 알고리즘을 써놓은 코드이며 의사코드는 다양한 스타일의 언어가 존재한다.

흐름도는 알고리즘을 표준화된 기호 및 도형으로 도식화 하여 데이터의 흐름과 수행되는 연산들의 순서를 표현하는 방법으로 순서도라고도 한다.

## 1.6 다양한 ‘프로그래밍 언어’ .....

포트란은 FORMula TRANslating system(수식 번역 시스템)의 약자이며 과학과 공학 및 수학적 문제들을 해결하기 위해 고안된 프로그래밍 언어이고 널리 보급된 최초의 고급 언어이다. 수학에서 사용하는 +, -, \*, /와 같은 사칙연산 기호와 sin, cos, tan의 삼각함수를 비롯하여 log, abs 등 다양한 수학함수들을 프로그램 내에서 그대로 사용하였다. 어셈블리 언어에 익숙해져 있던 1957년경, 포트란은 IBM에서 존 배커스(John Backus) 등의 전문가가 개발한 프로그래밍 언어 FORTRAN은 가장 오래된 언어지만 언어 구조가 단순해 놀라운 생명력을 갖추고 있어 지금도 과학 계산 분야 등에서는 많이 사용되고 있는 추세이다.

코볼(Common Business Oriented Language)은 협회 CODASYL이 1960년, 사무처리에 적합한 프로그래밍 언어로 개발한 언어이다. 포트란에 이어 두 번째로 개발된 고급 언어이며 대량의 데이터를 효율적으로 입력, 출력 및 처리하기 위해 개발되었다. 코볼 컴파일러만 있으면 어떠한 컴퓨터 기종이라도 코볼 프로그램을 작성하여 실행 가능하지만 코볼은 사무처리에 목적이 있으므로 다른 프로그래밍 언어에 비하여 파일이나 데이터베이스에서 데이터를 쉽게 읽고 쓰며, 또한 양식을 가진 보고서를 쉽게 만들 수 있는 등 사무처리에 효율적이라는 장점이 있다.

알골은 알고리즘(ALGOritm)을 표현하기 위한 언어로 ALGOritmic Language를 줄여서 만든 이름이다. 포트란이 미국을 중심으로 사용했다면 알골은 유럽을 중심으로 과학기술 계산을 프로그래밍 언어로 사용되었으며 알고리즘의 연구 개발에 적합한 언어로 개발되었다. 절차적 언어로서 구조화 프로그래밍에 적합하고, 최초로 재귀호출이 가능한 프로그래밍 언어인 알골은 파스칼, C 언어 등 이후 언어의 발전에 큰 영향을 주었으나, IBM이 주로 포트란을 사용하여 더 이상 대중화에 성공하지 못하였다.

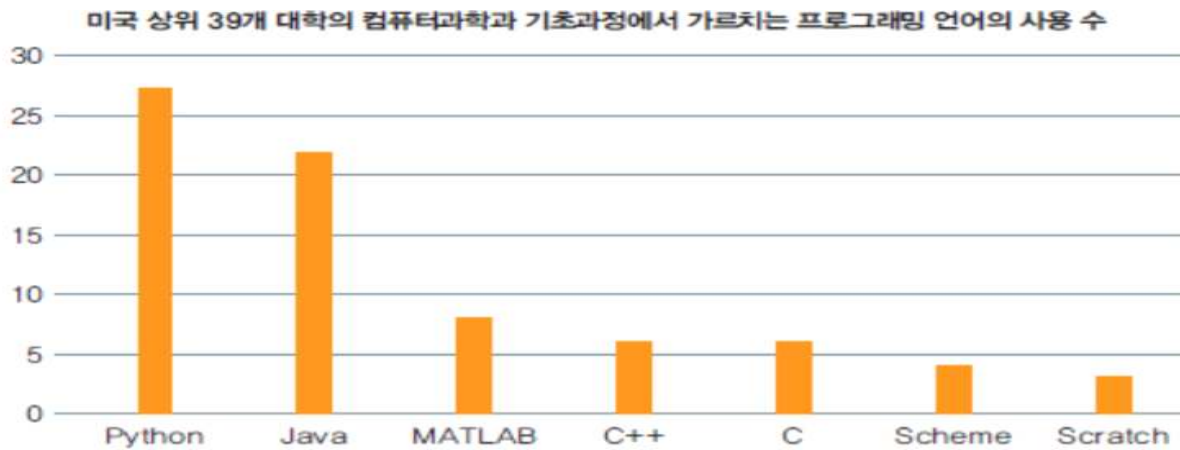
베이직(BASIC)은 ‘Beginner’s All-purpose Symbolic Instruction Code’(초보자의 다 목적용이고 부호를 사용하는 명령어 코드)의 약어이며 초보자도 쉽게 배울 수 있도록 만들어진 대화형 프로그래밍 언어이다. 1964년에 미국 다트머스(Dartmouth) 대학의 켄니(John Kemeny) 교수와 커즈(Thomas Kurtz) 교수가 개발하였고 대화형의 영어 단어를 바탕으로 약 200여 개의 명령어들로 구성된 가장 쉬운 대화형 프로그래밍 언어이다. 문장의 종류가 많지 않고 문법이 간단하며, 배우고 쓰기가 간단하고 쉽지만 컴파일 없이 바로 작동되나 인터프리터를 거쳐야 하므로 실행 속도가 느리다는 단점이 있다. 1980년대에 개인용 컴퓨터의 출현과 함께 베이직은 기본 개발 언어로 탑재되어 범용적인 언어로 널리 사용되었으며 마이크로소프트는 이 베이직을 기본으로 비주얼 베이직(Visual Basic)이라는 프로그램 언어를 개발하였다. 비주얼 베이직은 표준 베이직에 객체지향 특성과 그래픽 사용자 인터페이스를 추가한 프로그램 언어이자 통합개발환경이고 웹 프로그래밍 기술에서 많이 사용하는 언어 중 하나인 ASP(Active Server Page)도 VBScript를 사용하는데, VBScript도 베이직 문법을 그대로 사용하였다.

파스칼은 프랑스의 수학자인 파스칼(Pascal)의 이름에서 따온 언어이며 프로그램을 작성하는 방법인 알고리즘 학습에 적합하도록 1971년 스위스 취리히 공과대학교의 니콜라우스 비르트(Nicholas Wirth) 교수에 의해 개발된 프로그래밍 언어이다. 파스칼은 알골(Algol)을 모체로 해서 초보자들이 프로그램의 문법적 에러를 줄일 수 있도록 매우 엄격한 문법을 가진 프로그래밍 언어이며 구조적인 프로그래밍(structured programming)이 가능하도록 begin~end의 블록 구조가 설계되었다. 1980년에서 1990년대까지 대부분의 대학에서 프로그래밍 언어의 교과과정으로 파스칼을 채택되었으며 애플사는 1980년 초, 파스칼 문법에 객체지향 기능을 추가시킨 오브젝트 파스칼(Object Pascal) 개발하였다. 1980년대에는 볼랜드사에서 파스칼을 발전시켜 터보 파스칼(Turbo Pascal)이라는 제품으로 상용화하였고 볼랜드사는 1990년 중반에 마이크로소프트사의 비주얼 베이직과 유사한 오브젝트 파스칼 언어를 기반으로 하는 델파이(Delphi)를 출시하였다.

C++은 1972년에 개발된 C 언어가 1983년에 프로그램 C++언어 로 발전되었으며 C++는 객체지향 프로그래밍(OOP: Object Oriented Programming)을 지원하기 위해 C언어가 가지는 장점을 그대로 계승하면서 객체의 상속성(inheritance) 등의 개념을 추가한 효과적인 언어이다. AT&T의 얀 스트로스트룹(Bjarne Stroustrup)이 개발하였고 C++언어가 C언어와 유사한 문법을 사용함으로써 C언어에 익숙한 프로그래머들이 C++언어를 쉽게 배울 수 있다는 장점을 갖고 있는 언어이다.



파이썬은 현재 미국의 대학에서 컴퓨터 기초 과목으로 가장 많이 가르치는 프로그래밍 언어 중 하나이며 1991년 네덜란드의 귀도 반 로섬(Guido van Rossum)이 개발한 객체지향 프로그래밍 언어이다. 비영리의 파이썬 소프트웨어 재단이 관리하는 개방형, 공동체 기반 개발 모델 C#으로 구현된 닷넷프레임워크 위에서 동작하는 IronPython, Java로 구현되어 JVM위에서 돌아가는 Jython, 파이썬 자체로 구현된 PyPy 등 다양한 언어로 만들어진 버전 등 다양하다. C언어로 구현된 C 파이썬(cpython) 구현이 사실상의 표준이며 프로그래밍 언어 파이썬이 대학의 컴퓨터기초 교육에 많이 활용되었고 무료이며, 간단 하면서 효과적으로 객체지향을 적용할 수 있는 강력한 프로그래밍 언어라고 할 수 있다. 베이직과 같은 인터프리터 언어로 간단한 문법구조를 가진 대화형 언어이므로 쉽고 빠르게 개발할 수 있어, 개발기간이 매우 단축되는 것이 장점이다.



자바는 1990년 양방향 TV를 만드는 제어 박스의 개발을 위한 그린 프로젝트(Green Project)를 시작으로 초기에는 객체지향 언어로 광범위하게 이용되고 있는 C++ 언어를 이용하였고 다양한 하드웨어를 지원하는 분산 네트워크 시스템 개발에 부족함을 느낀 개발팀은 C++ 언어를 기반으로 오크(Oak, 떡갈나무)라는 언어를 직접 개발하였다. 제임스 고슬링(James Gosling)은 이 오크라는 언어를 발전시켜 자바라는 범용적인 프로그래밍 언어를 개발하였다.

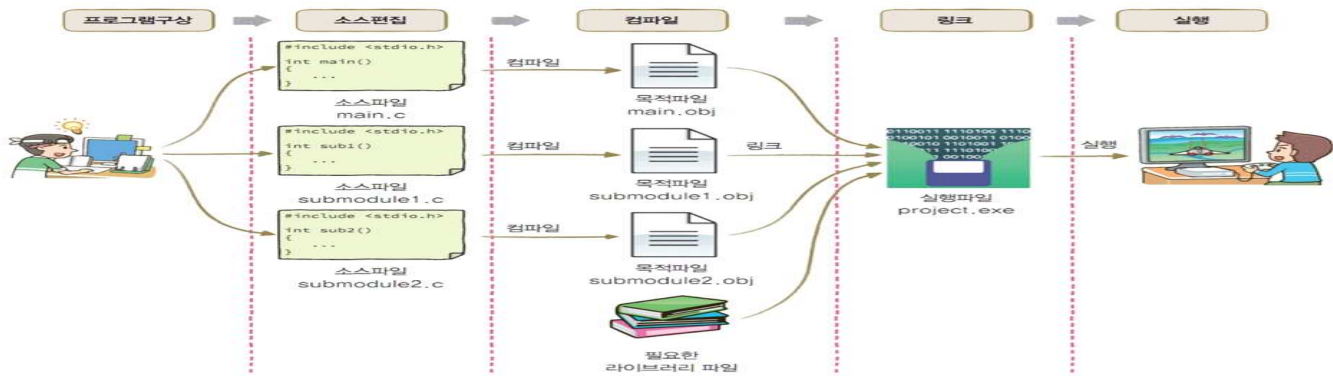


스크래치는 2007년 MIT 대학의 미디어랩(Media Lab)에서 개발한 비주얼 프로그래밍(visual programming) 개발도구이며 브라우저에서 직접 개발하는 환경으로 커뮤니티 기반 웹 인터페이스로 구성되어있다.

일반인과 청소년 또는 프로그래밍 초보자를 학생들을 대상으로 컴퓨터 프로그래밍의 개념을 이해할 수 있도록 도와주는 교육용 프로그래밍 언어(educational programming language)이므로 다양한 이미지나 사운드를 제공하여 쉽게 사용 가능하며 직관적으로 누구나 쉽게 이해할 수 있는 블록을 끼워 맞춰 프로그램을 작성하는 형식으로 개발이 가능하다.

엘리스는 카네기 멜론 대학교에서 ‘마지막 강의’로 알려진 랜디 포시(Randy Pausch) 교수가 주도하여 개발한 프로그래밍 교육 도구이며 쉽게 삼차원 인터랙티브 그래픽 콘텐츠를 제작하면서 객체지향 프로그래밍 기법을 학습할 수 있는 교육용 소프트웨어이다. 엘리스는 컴퓨터 전공학과와 저학년 학생들이 프로그래밍을 어렵게 생각하고 포기하지 않도록 프로그래밍 개념과 실습을 학습할 수 있도록 지원하는 프로그래밍 소프트웨어이며 특히 프로그래밍 언어 자바, C++, C#과 같은 객체지향 프로그래밍 기법을 학습하는데 유용한 개발 도구라 할 수 있다. 다양한 캐릭터나 객체를 끌어다 놓는(Drag & Drop) 방식으로 프로그래밍하며 비주얼 프로그래밍 기법은 프로그래밍언어 문법학습에 대한 학습자들의 부담을 덜어 주고 쉽게 프로그래밍이 가능하다. 클래스와 객체로 콘텐츠 제작에 사용할 수 있는 다양한 캐릭터를 제공하고, 사용법이 쉬우므로 일반인도 쉽게 삼차원 애니메이션이나 게임을 제작할 수 있다.

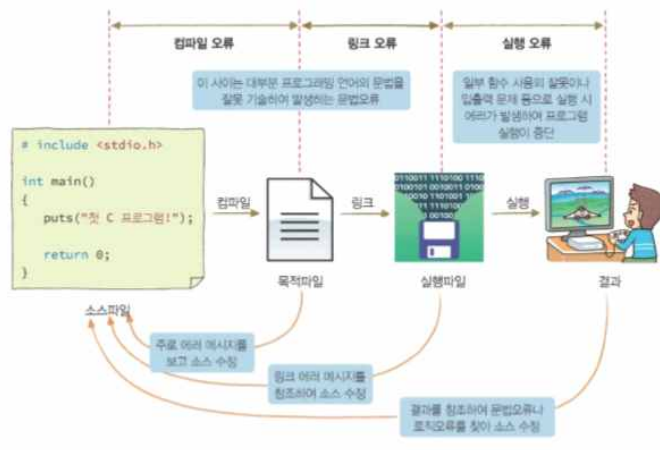
## 2.1 프로그램 구현 과정과 통합개발환경 .....



프로그램을 구현하기 위해서는 프로그램구상, 소스편집, 컴파일, 링크, 실행의 5단계가 존재하며 간단한 프로그램은 하나의 소스파일로 구성되어 있고 프로그램이 커진다면 여러 개의 소스파일로 구성하는 것이 효율적이다.

프로그램 구상과 소스편집 단계에서 소스코드는 선정된 프로그래밍 언어인 C 프로그램 자체로 만든 일련의 명령문을 의미하며 소스파일(source file)이란 C와 같은 프로그래밍 언어로 원하는 일련의 명령어가 저장된 파일이라고 할 수 있다. 컴파일 단계는 소스파일에서 기계어로 작성된 목적파일(object file)을 만들어내는 과정이고 컴파일러에 의해 처리되기 전의 프로그램을 소스코드 (source code)라고 하며 컴파일러에 의해 기계어로 번역된 프로그램은 목적코드(object code)라고 한다. 링커란 하나 이상의 목적파일과 라이브러리를 포함시켜 하나의 실행파일(execute file)로 만들어 주는 프로그램이며 라이브러리는 개발환경에서 미리 만들어 컴파일해 저장해 놓는데, 이 모듈을 라이브러리(library)라 칭한다.

오류 또는 에러란 프로그램 개발 과정에서 나타나는 문제이며 발생시점에 따라 분류를 할 수 있는데 첫 번째로는 오류를 수정하기가 비교적 쉬운 컴파일 오류, 두 번째로는 주로 함수 이름을 잘못 기술하여 발생하는 링크 오류, 마지막으로 실행하면서 오류가 발생하여 실행을 중지되는 경우인 실행오류로 총 세가지의 오류로 분류할 수 있다. 또한 오류의 원인과 성격에 따라 분류도 가능하다. 오류의 원인과 성격에 따라 분리하면 자주 발생하며 주로 프로그래밍 언어를 잘못 기술하여 발생하는 문법 오류 (syntax error)와 내부 알고리즘이 잘못되거나 원하지 않는 결과가 나오는 등의 오류인 논리오류 (logic error) 두 가지로 분류 할 수 있다.



디버깅(debugging)이란 프로그램 개발 과정에서 발생하는 오류를 찾아 소스를 수정하여 다시 컴파일, 링크, 실행을 순차적으로 하는 과정이다.

디버거(debugger)는 디버깅을 도와주는 프로그램이다. 여기서 벌레라는 단어의 버그(bug)란 바로 오류를 뜻하는 단어를 말한다.

통합 개발 환경인 IDE(Integrated Development Environment)는 프로그램 개발에 필요한 편집기(editor), 컴파일러(compiler), 링커(linker), 디버거 (debugger) 등을 통합하여 편리하고 효율적으로 제공하는 개발환경이며 대표적으로는 프로그램 언어 C/C++ 뿐만 아니라 C#, JavaScript, Python, Visual Basic 등 여러 프로그램 언어를 이용할 수 있게 다양한 프로그래밍 언어와 환경을 지원하는 통합개발환경인 마이크로소프트사의 비주얼스튜디오와 PDE(Plug-in Development Environment) 환경을 지원하여 확장이 가능한 통합개발환경인 이클립스 C/C++ 개발자용 IDE 가 있다.

2.2 비주얼 스튜디오 설치와 C프로그램의 첫 개발 .....

비주얼 스튜디오 설치방법을 쉽게 설명하기 위해 아래와 같이 사진을 첨부하였으니 보고 따라하면 금방 설치가 될 것이다.

다운로드

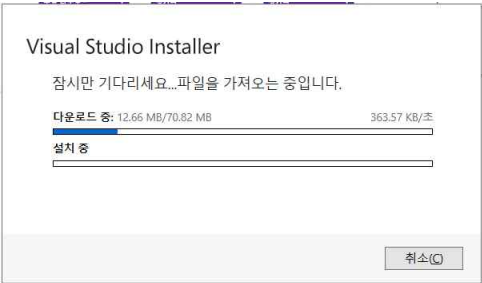
Step 01

<https://visualstudio.microsoft.com/ko/downloads/>에 접속후 무료 다운로드를 클릭하면 설치를 시작한다.



Step 02

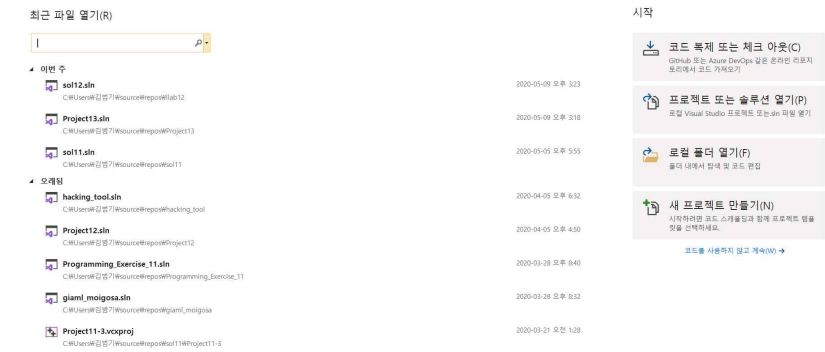
시작 전에 설치를 구성할 수 있도록 몇가지 항목을 선택해야 된다는 문구가 뜨면 “계속” 버튼을 눌러 진행해준다.



Step 03

워크로드에서 필요한 개발패키지를 선택할 수 있는데 우리는 c언어를 사용할 것 이므로 c++용 사용한 데스크톱 개발을 체크한 뒤 설치해준다.

Visual Studio 2019



Step 04

설치가 완료되면 다음과 같이 프로젝트를 생성할 수 있는 창이 뜨는데 새 프로젝트 만들기를 클릭하여 C언어 개발을 시작해 볼 수 있다.

책에 나온 예제를 풀어보기 전 유의할 점이 있다. C소스는 영문자의 대소문자를 구분하며 #, <, >, (, ), ;, {, }와 같은 특별한 의미의 여러 문자들로 구성 되어 있다. 함수 main()은 대소문자로 구분하여 기술하고 중간에 공백이 들어갈 수 없으며 소괄호 ( ) 와 중괄호 {}는 구분하고 컴파일러는 하나의 오타도 허용하지 않으므로 편집기에 주의를 기울여서 행과 열을 맞춰 정확히 소스를 입력해야 한다. 또한 솔루션이란 여러개의 프로젝트로 구성되어 있으며 프로젝트에 소스파일을 추가하여 C언어를 통한 프로그래밍을 할 수 있으며 각 프로젝트마다 main() 함수가 있는 소스는 단 하나 존재한다.

## 새 프로젝트 만들기

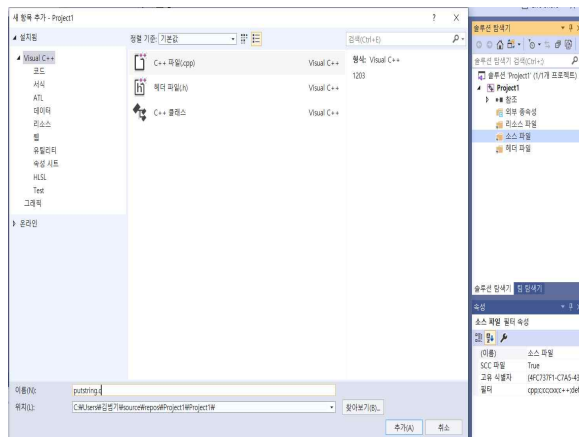


솔루션을 만들기 위해 새 프로젝트를 만들기 선택한 후 빈 프로젝트를 클릭하고 다음 버튼을 누른다.

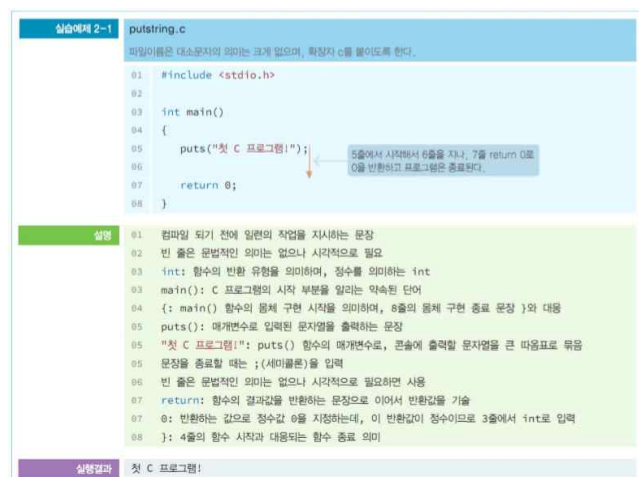
## 새 프로젝트 구성



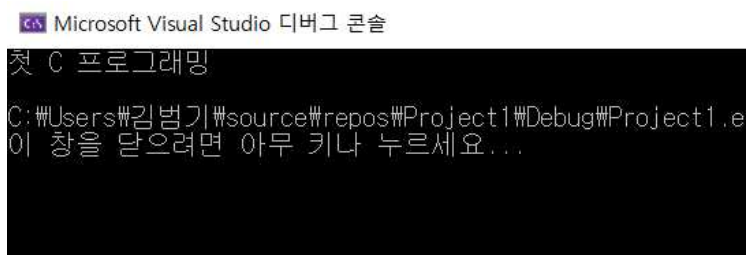
새 프로젝트 구성의 입력란에 원하는 프로젝트의 이름, 위치, 솔루션의 이름을 입력한 뒤 만들기 버튼을 클릭하여 새 프로젝트를 생성한다.



프로젝트가 생성되었으면 소스파일을 우클릭 한 후 추가 -> 새 항목 -> C++파일 선택 후 해당 소스파일의 이름을 입력한 뒤 확장자 .C를 붙여주어 새로운 C 소스파일을 생성한다.



책에 나온 예제 2-1번의 소스를 입력 후 컴파일 및 실행을 하면 다음과 같은 결과가 나온다.



## 2.3 C프로그램의 이해와 디버깅 과정 .....

C프로그램의 시작과 끝은 모두 함수이며 C프로그램과 같은 절차지향 프로그램은 함수(function)로 구성되어 있다. 함수 하나하나가 프로그램 단위라고 할 수 있으며 함수는 'a, b, c...'와 같은 입력(input)을 받아와서 'y'와 같은 결과(output) 값을 만들어 내는 기계장치와 유사하다. '입력'은 여러 개 사용될 수 있지만 결과 값은 꼭 하나여야 하며 프로그래머가 직접 만드는 함수인 사용자 정의 함수(user defined function)와 시스템에 미리 정의되어 있는 함수인 라이브러리 함수(library function)로 구분할 수 있다.

함수에서도 다양한 용어가 있는데 처음으로 사용자 정의 함수를 만드는 과정인 함수 정의(function definition) 두 번째로 라이브러리 함수를 포함해서 만든 함수를 사용하는 것을 의미하는 함수 호출(function call), 세 번째로 함수를 정의할 때 나열된 여러 입력 변수인 매개변수(parameters) 마지막으로 함수 호출 과정에서 전달되는 여러 입력값인 인자(argument)가 대표적인 용어이다.

시작함수 main() 정의의 첫 줄에 int와 void가 들어가며 각각 함수가 자신의 작업을 모두 마친 후 반환하는 값의 유형이고 함수로 값을 전달할 때 필요한 입력 형식은 { ... } 중괄호이다. '{' 와 '}'를 사용하여 함수의 기능을 구현하며 함수 main()이 실행되는 과정은 먼저 프로그램이 실행되면 운영체제는 프로그램에서 가장 먼저 main()함수를 찾고 입력 형태의 인자로 main() 함수를 호출한다. 호출된 main()함수의 첫 줄을 시작으로 마지막 줄까지 실행하면 프로그램은 종료한다. 만일 main() 함수 내부에서 puts()와 같이 라이브러리 함수를 호출 및 인자 "Hello World!"를 전달한다면 puts()를 실행한 후 다시 main()으로 돌아와 그 다음 줄인 return 0;을 실행한다.

라이브러리 함수 puts()와 printf()를 사용하려면 #include는 바로 뒤에 기술하는 헤더파일 stdio.h를 삽입해야하며 코드로는 **#include <stdio.h>**로 나타낼 수 있다.

puts()함수는 원하는 문자열을 괄호 ("원하는 문자열") 사이에 기술하며 인자를 현재 위치에 출력한 후 다음 줄 첫 열로 이동하여 출력을 기다리는 함수이다. 괄호 사이에 아무것도 없을 시 인자가 없으므로 오류가 발생하지만 puts("")와 같이 공백 문자열을 입력하면 현재 출력 위치에 공백 문자열을 출력한 후 다음 줄로 이동하는 엔터키와 비슷한 효과를 낼 수 있다.

함수 printf()는 원하는 문자열을 괄호 ("원하는 문자열") 사이에 기술하며 printf("")와 같이 공백 문자열을 인자로 전달하면 현재 위치에 공백문자를 출력하지만 결과는 아무것도 출력되는 것이 없다. 함수 호출 printf("\n")와 같이 기술하면 출력 위치를 새로운 줄 첫 열로 이동하게 하는 효과가 있어 인자인 문자열을 출력하고 다음 줄로 이동하여 출력 위치를 지정하기 위해 주로 활용한다. 아무것도 출력 없이 출력 위치를 다음 줄로 이동하기 위해서는 함수 puts("") 또는 함수 printf("\n")로 호출하여 사용할 수 있다.

### 예제 2-3

실습예제 2-3

printmline.c

```
01 #include <stdio.h>
02
03 int main(void)
04 {
05     puts("세 번째 C 프로그램!");
06     puts("");
07     printf("-- 비주얼 스튜디오 C 프로그래밍 과정 --\n");
08     printf("\n");
09     puts("1. 솔루션과 프로젝트 만들기");
10     printf("2. 소스파일 편집\n");
11     printf("3. 실행\n");
12
13     return 0;
14 }
```

결과 :

Microsoft Visual Studio 디버그 콘솔

세번째 C 프로그램!

—비주얼 스튜디오 C 프로그래밍 과정—

1. 솔루션과 프로젝트 만들기  
2. 소스파일 편집  
3. 실행

C:\Users\김범기\source\repos\Project1\WD  
이 창을 닫으려면 아무 키나 누르세요...



### 3.1 프로그래밍 기초 .....

솔루션은 여러개의 프로젝트를 가지며, 다시 프로젝트는 여러 소스파일을 포함한 여러 자원(source)으로 구성된다. 비주얼 스튜디오의 한 프로젝트는 단 하나의 함수 main()과 다른 여러 함수로 구현되며, 최종적으로 프로젝트이름으로 하나의 실행 파일이 만들어진다.

이러한 단어가 C에서 사용되는 기본 키워드로 문법적인 고유한 의미가 있다.



auto	do	goto	signed	unsigned
break	double	if	sizeof	void
case	else	int	static	volatile
char	enum	long	struct	while
const	float	return	typedef	
default	for	short	union	

키워드는 문법적으로 고유한 의미를 갖는 예약된 단어를 뜻하며 예약어(reserved word)라고도 한다. C프로그램에서는 미국표준화위원회(ANSI: American National Standard Institute)에서 지정한 32개의 기본적인 단어들이다. 또한 비주얼 스튜디오 편집기에서 키워드는 기본적으로 파란색으로 표시한다.

식별자는 프로그래머가 자기 마음대로 정의해서 사용하는 단어로 변수이름, age, yea, 함수이름, puts, main, printf 등이 식별자라고 할 수 있다. 식별자는 사용 시 영문자(대소문자 알파벳), 숫자(0 ~ 9), 밑줄(\_)로 구성되어야하며 식별자의 첫 문자로 숫자가 나올 수 없다. 또한 프로그램 내부의 일정한 영역에서는 서로 구별해야하며 당연히 키워드는 식별자로 이용할 수 없다. 식별자는 대소문자를 모두 구별하는데 예를들면 변수 Count, count, COUNT는 모두 다른 변수인 것이다. 식별자의 중간에 공백(space)문자가 들어갈 수 없다.

컴퓨터에게 명령을 내리는 최소 단위를 문장(statement)이라고 한다. 문장은 마지막에 세미콜론 ;으로 종료하며 문장 마지막에 ;을 빠뜨리면 컴파일 시간에 문법 오류가 발생하니 주의할 필요가 있다. 여러 개의 문장을 묶으면 블록(block)이라고 하며 { 문장1, 문장2, ... } 처럼 중괄호로 열고 닫아서 표기한다. 들여쓰기는 블록 내부에서 문장들을 탭(tab) 정도만큼 오른쪽으로 들여 쓰는 소스 작성 방식이며 적절한 줄 구분과 빈 줄 삽입, 그리고 들여쓰기는 프로그램의 이해력을 돕는데 매우 중요한 요소이다.

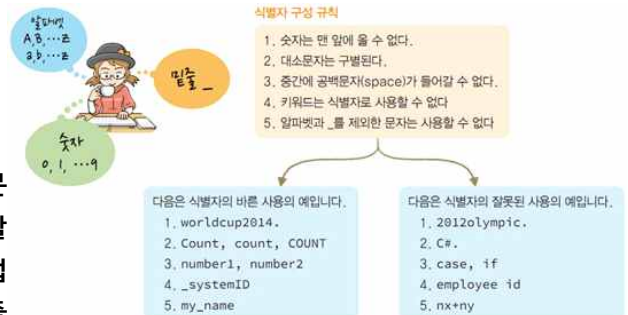


그림 3-5 식별자 구성 규칙과 사용 예

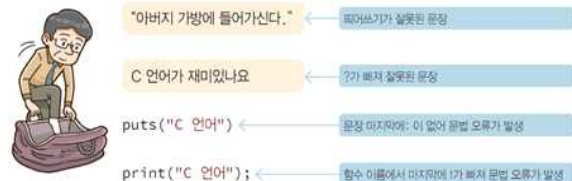


그림 3-6 문장과 문법 오류

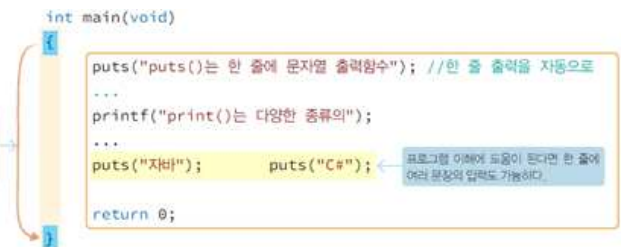


그림 3-7 블록과 들여쓰기의 이해

주석은 타인은 물론이거니와 자신을 위해서라도 반드시 필요하며, 주석에는 자신을 비롯한 이 소스를 보는 모든 사람이 이해할 수 있도록 도움이되는 설명을 담고 있어야한다. 잘 처리된 주석이란 시각적으로 정돈된 느낌을 주어야 하며, 프로그램의 내용을 적절히 설명해주어야 한다. 주석은 2가지의 처리방법이 있는데 먼저 한 줄 주석 //는 // 이후부터 그 줄의 마지막까지 주

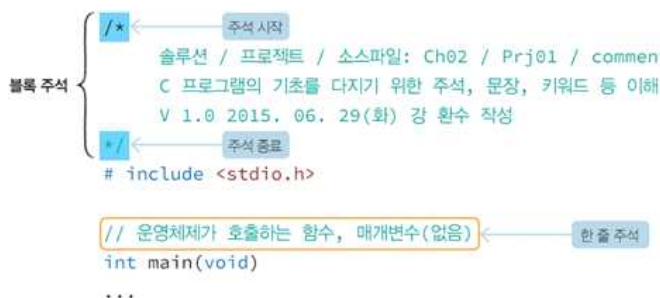


그림 3-10 블록 주석과 한 줄 주석 예

석으로 인식하여 현재 줄의 처음이나, 문장 뒤부터 중간에서의 주석은 주로 한 줄 주석을 이용한다. 블록 주석 /\* ... \*/은 여러 줄에 걸쳐 설명을 사용할 때 이용하며 주석 시작은 /\*로 표시하고, 종료는 \*/로 표시한다. 프로그램의 처음 부분에는 주로 여러 줄에 걸친 블록 주석을 사용하며 작성자와 소스의 목적과 프로그램의 전체적 구조와저작권 정보 등 파일 관련 정보들을 작성한다.



## 3.2 자료형과 변수선언



그림 3-13 자료형과 식재료의 분류

C프로그래밍 언어에서 다루는 다양한 자료도 기본형(basic types), 유도형(derived types), 사용자정의형(user defined types) 등으로 나눌 수 있으며, 기본형은 다시 정수형, 실수형, 문자형, void로 나뉘는데, 프로그래머는 이러한 자료에 적당한 알고리즘을 적용해 프로그램을 작성한다. 즉 자료형(data type)은 프로그래밍 언어에서 자료를 식별하는 종류를 말한다.

저장 공간을 변수(variables)라 부르며 변수에는 고유한 이름이 붙여지고 물리적으로 기억장치인 메모리에 위치한다. 용기에 다양한 식재료를 담듯이 변수에 여러 값을 저장할 수 있고 저장되는 값에 따라 변수값은 바뀔 수 있으며 마지막에 저장된 하나의 값만 저장을 유지한다. 요리에서 프라이팬이나 조리용 그릇처럼 프로그래밍에도 정수와 실수, 문자 등의 자료값을 중간 중간에 저장할 공간이 필요하다.

변수선언은 컴파일러에게 프로그램에서 사용할 저장 공간인 변수를 알리는 역할을 하며 변수는 고유한 이름이 붙여지고, 자료값이 저장되는 영역이다. 자료형을 지정한 후 변수이름을 나열하여 표시하며 int, double, float와 같이 자료형 키워드를 사용한다. 변수 이름은 관습적으로 소문자를 이용하고 반드시 변수선언 이후라야 값을 저장하거나 참조가 가능하다.

```
int math = 99, korean = 90, science = 94;
```

```
int math = 99;
int korean = 90;
int science = 94;
```

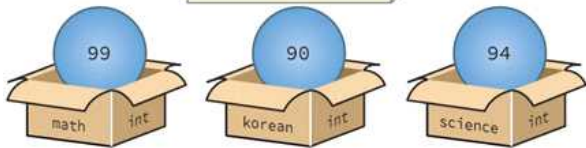


그림 3-23 여러 변수 선언과 초기화

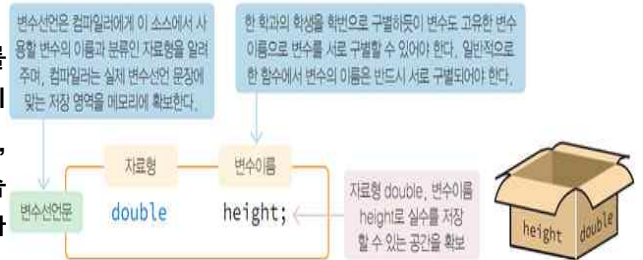


그림 3-16 자료형을 이용한 변수선언



그림 3-19 변수선언과 대입문

대입문은 원하는 자료값을 선언된 변수에 저장한다. 대입연산자(assignment operator)표시인 '='를 사용하며 오른쪽에 위치한 값을 이미 선언된 왼쪽 변수에 저장한다라는 의미이다. 대입문(assignment statement)은 변수명 age를 int 형으로 선언한 후, 변수 age에 20을 저장하는 문장이며 가장 마지막에 저장된 값만이 남는다.

초기값을 저장할 때 변수를 선언하면서 변수명 이후에 대입연산자 =와 수식이나 값이 오면 바로 지정한 값으로 초기값이 저장된다. 오류가 발생하는 경우에는 변수를 선언만하고 자료값을 아무것도 저장하지 않으면 원치 않는 값이 들어 있어서 오류가 나며 초기값이 없는 변수를 사용하더라도 오류가 발생한다.

변수(variables)에서 주요 정보인 변수이름, 변수의 자료형, 변수 저장 값을 변수의 3요소라 한다. 즉 변수선언 이후 저장값이 대입되면 변수의 3요소가 결정된다. 문장에서 변수의 의미는 저장공간 자체와 저장공간에 저장된 값으로 나눌 수 있다.

대입에서 l-value와 r-value는 대입연산자 =의 왼쪽에 위치하는 변수를 lvalue 또는 l-value라 하며

l-value는 반드시 수정이 가능한 하나의 변수이어야 하고 r-value는 l-value에 저장할 자료값을 반환하는 표현식(상수, 변수, 수식, 함수...)이 있다. 대입 연산자 =의 왼쪽에 위치한 변수는 저장공간 자체의 사용을 의미하며 대입 연산자 =의 오른쪽에 위치한 변수는 저장 값의 사용을 의미한다.

```
l-value = r-value;
```

```
21 = 20 + 1; //오류발생 오류: 식이 수정할 수 있는 lvalue여야 합니다.
```

그림 3-21 l-value와 r-value

```
int math = 99;
int korean = 90;
int science;

int total = math + korean + science;
```

그림 3-24 초기화 되지 않은 지역변수의 사용에서 컴파일 오류

### 3.3 기본 자료형 .....

C의 자료형은 기본형(basic data types), 유도형(derived data types), 사용자정의형(user defined data types)으로 나뉘며 기본이 되는 자료형으로는 다시 정수형, 실수형(부동소수형), 문자형, 무치형 ,무치형 자료형인 void가 있으며 void는 아무런 자료형도 지정하지 않은 자료형이며 함수의 인자 위치에 놓이면 ‘인자가 없다’라는 의미로 사용되고 함수의 반환값에 놓으면 ‘반환값이 없다’라는 의미로 사용된다. 유도형은 배열(array), 포인터(pointer), 함수(function) 등으로 구성되어 있으며 사용자정의형은 기본형과 유도형을 이용하여 프로그래머가 다시 만드는 자료형으로 열거형(enumeration), 구조체(structure), 공용체(union)로 분류된다.

정수형(integer types)의 기본 키워드인 int는 십진수, 팔진수, 십육진수의 정수가 다양하게 저장되며 파생된 자료형인 short와 long은 short, short int, long, long int이 있다. 키워드 signed에서 signed 키워드는 생략이 가능하며 signed int와 int는 같은 자료형을 뜻한다. 키워드 unsigned는 0과 양수만을 처리하며 short, int, long 앞에 표시하여 사용한다. 정수형 저장공간은 비주얼 스튜디오에서 short는 2바이트 int, long은 모두 4바이트의 크기를 가지고 있다.



그림 3-31 정수 자료형의 저장공간 크기

`float x = 3.14;` //float x = 3.14;인 경우, 경고 발생  
 warning C4305: '초기화 중': 'double'에서 'float'(으)로 줄입니다.  
 그림 3-33 float 형 변수에 부동소수 상수로 저장한 경우의 경고

실수형 키워드는 float, double, long double 세 가지가 있으며 비주얼 스튜디오에서 float는 4바이트이며, double과 long double은 모두 8바이트이다. 또한 소수 3.14와 같은 표현은 모두 자료형 double로 인식하며 float형 변수에 저장하면 컴파일 경고나 오류가 발생한다.

표 3-7 기본 자료형의 저장공간 크기와 표현범위(자료형에서 []은 생략 가능함)

분류	자료형	크기	표현범위
문자형	char	1 바이트	-128(-2 <sup>7</sup> ) ~ 127(2 <sup>7</sup> -1)
	signed char	1 바이트	-128(-2 <sup>7</sup> ) ~ 127(2 <sup>7</sup> -1)
	unsigned char	1 바이트	0 ~ 255(2 <sup>8</sup> -1)
정수형	[signed] short [int]	2 바이트	-32,768(-2 <sup>15</sup> ) ~ 32,767(2 <sup>15</sup> -1)
	[signed] [int]	4 바이트	-2,147,483,648(-2 <sup>31</sup> ) ~ 2,147,483,647(2 <sup>31</sup> -1)
	[signed] long [int]	4 바이트	-2,147,483,648(-2 <sup>31</sup> ) ~ 2,147,483,647(2 <sup>31</sup> -1)
	[signed] long long [int]	8 바이트	9,223,372,036,854,775,808(-2 <sup>63</sup> ) ~ 9,223,372,036,854,775,807(2 <sup>63</sup> -1)
	unsigned short [int]	2 바이트	0 ~ 65,535(2 <sup>16</sup> -1)
정수형	unsigned [int]	4 바이트	0 ~ 4,294,967,295(2 <sup>32</sup> -1)
	unsigned long [int]	4 바이트	0 ~ 4,294,967,295(2 <sup>32</sup> -1)
	[unsigned] long long [int]	8 바이트	0 ~ 18,446,744,073,709,551,615(2 <sup>64</sup> -1)
부동소수형	float	4 바이트	대략 10 <sup>-38</sup> ~ 10 <sup>38</sup>
	double	8 바이트	대략 10 <sup>-308</sup> ~ 10 <sup>308</sup>
	long double	8 바이트	대략 10 <sup>-308</sup> ~ 10 <sup>308</sup>

C 언어에서 문자형 자료공간에 저장되는 값은 실제로 정수값이며 아스키 코드 표에 의한 값이다. 아스키 코드(ASCII: American Standard Code for Information)에서 소문자 ‘a’는 16진수로 61, 이진수로는 1100001, 십진수로 97로 표기한다. 다음 표는 기본 자료형의 저장공간 크기와 표현 범위이다.

연산자 sizeof는 자료형, 변수, 상수의 저장공간 크기를 바이트 단위 반환하며 자료형 키워드로 직접 저장공간 크기를 알려면 자료형 키워드에 괄호가 반드시 필요하다.



그림 3-36 오버플로 발생 원리

자료형 unsigned char는 8비트로 0에서 255까지 저장 가능하며 만일 256을 저장하면 0으로 저장한다. 정수의 순환에서 정수형 자료형에서 최대값+1은 오버플로로 인해 최소값이 저장되며 마찬가지로 최소값-1은 최대값이다. 실수형 float 변수에 정밀도가 매우 자세한 수를 저장하면 언더플로 (underflow)가 발생하여 0이 저장된다.

3.4 상수 표현방법

상수(constant)는 이름 없이 있는 그대로 표현한 자료값이다.

표 3-8 상수의 종류

구분	표현 방법	설명	예
리터럴 상수 (이름이 없는 상수)	정수형 실수형 문자 문자열 상수	다양한 상수를 있는 그대로 기술	32, 025, 0xf3, 10u, 100L, 30LL, 3.2F, 3.15E3, 'A', '\n', '\0', '\24', '\x2f', "C 언어", "프로그래밍 언어\n"
심볼릭 상수 (이름이 있는 상수)	const 상수 매크로 상수 열거형 상수	키워드 const를 이용한 변수 선언과 같으며, 수정할 수 없는 변수 이름으로 상수 정의 전처리기 명령어 #define으로 다양한 형태를 정의 정수 상수 목록 정의	const double PI = 3.141592;  #define PI 3.141592  enum bool {FALSE, TRUE};

우린 생활에서 숫자 32, 32.4, 문자 \*, &, # 그리고 문자열 "Hello World!"등을 사용한다. 리터럴 상수는 소스에 그대로 표현해 의미가 전달되는 다양한 자료값을 말하며 10, 24.3과 같은 수, "C는 흥미롭습니다."와 같은 문자열을 뜻한다. 심볼릭 상수는 변수처럼 이름을 갖는 상수를 말하며 심볼릭 상수를 표현하는 방법은 const 상수(const constant), 매크로 상수(macro constant)가 있다.

리터럴 상수는 정수, 실수, 문자, 문자열 상수가 있고 문자 상수를 표현시에는 문자 하나의 앞 뒤에 작은따옴표(single quote)를 넣어서 표현한다. 코드값이 97인 문자 'a'이다. 함수 printf()로 문자 상수를 출력하려면 %c 또는 %C 사용해야 하며 %c의 c는 문자 character를 의미한다.

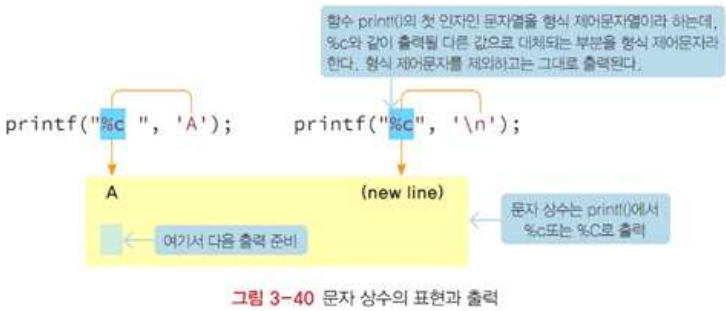


그림 3-40 문자 상수의 표현과 출력

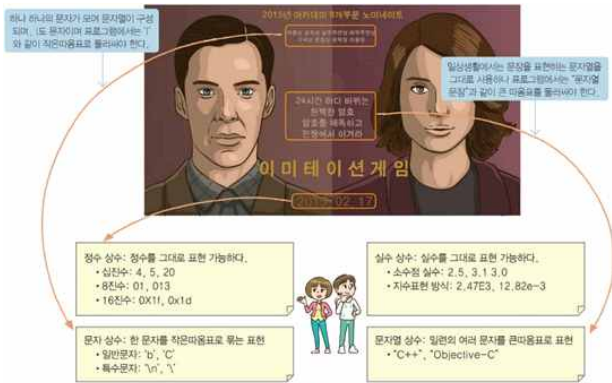


그림 3-39 리터럴 상수의 종류와 표현방법

이스케이프 문자란 \n와 같이 역슬래쉬 \와 문자의 조합으로 표현하는 문자이며 '\n'이 새로운 줄(new line)을 의미하는 대표적인 이스케이프 문자이다. 또한 문자열에도 사용이 가능하고 이스케이프 문자는 제어문자, 특수문자 또는 확장문자라고도 부른다.

정수형 리터럴 상수의 다양한 형태는 다음과 같이 100L, 20U, 5000UL이 있는데 정수 뒤에 l 또는 L을 붙이면 long int u 또는 U는 unsigned int ul 또는 UL은 unsigned long을 뜻하며 long long형은 LL, ll과 ULL, ull로 표기한다. 십육진수 표현방식에서 상수의 정수표현은 십진수로 인식하며 숫자 0을 정수 앞에 놓으면 팔진수(octal number)로 인식하고 숫자 0과 알파벳 0x, OX를 숫자 앞, 십육진수(hexadecimal number)로 인식한다. 십육진수는 0에서 9까지의 수와 알파벳 a, b, c, d, e, f(대소문자 모두 가능)하고 함수 printf()에서 정수를 출력할 때 %d를 사용하는데 이때 d는 십진수라는 decimal의 앞글자인 d를 뜻한다. 지수표현 방식에서 3.14E+2는 3.14\*102를 말하며 함수 printf()에서 지수표현 방식과 함께 일반 실수를 출력할때는 %lf의 형식 제어문자를 사용하고 형식 제어문자 %lf로 출력되는 실수는 소수점 6자리까지 출력한다. 실수형 리터럴 상수에서 실수형 상수도(float, doublelong double) 기본은 double 상수 float은 상수 숫자 뒤에 f나 F를 붙인다. long double 상수 숫자 뒤에는 L 또는 l을 붙여 표시한다. 키워드 const는 변수로는 선언되지만 일반 변수와는 달리 초기값을 수정할 수 없는 심볼릭 상수이며 상수는 변수선언 시 반드시 초기값을 저장하여야 한다.상수는 다른 변수와 구별하기 위해 관례적으로 모두 대문자로 선언하며 변수 RATE는 상수로 선언하는 구문이다. 선언 이후 저장값을 수정하기 위해 대입하면 문장에서 컴파일오류 C2166이 발생한다.



그림 3-45 키워드 const의 위치



그림 3-46 심볼릭 상수 선언과 오류



## 4.1 전처리 .....

전처리기 역할은 C에서는 컴파일(compile) 전에 전처리기(preprocessor)의 전처리(preprocess) 과정이 필요한데 결과인 전처리 출력파일을 만들어 컴파일러에게 보내는 작업을 수행하는 역할을 하는 것이 전처리기이다. 전처리 지시자(preprocess directives)는 #include, #define과 같이 전처리 지시자는 항상 #으로 시작하며 마지막에 세미콜론 ; 이 없는 등 일반 C 언어 문장과는 구별되어있다. 또한 조건 지시자로 #if, #elif, #else, #endif, #ifdef, #ifndef, #undef 등이 있다.

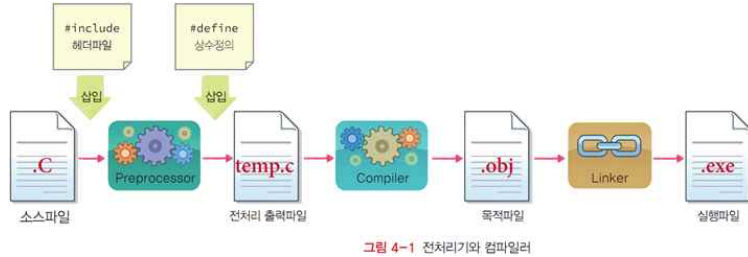


그림 4-1 전처리기와 컴파일러

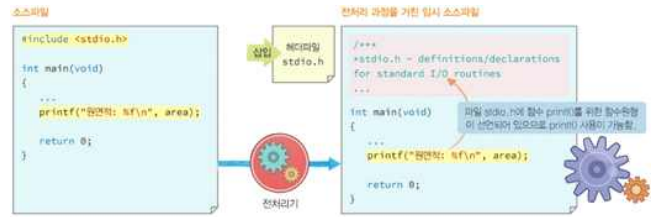


그림 4-4 전처리기와 컴파일러

#include <stdio.h>, #include, #define 등의 헤더파일은 자료형의 재정의 typedef, 함수원형(prototype) 정의 등과 같은 문장이 있는 텍스트 파일이며 대표적인 헤더파일은 확장자 \*.h를 사용하는 stdio.h가 있다.

```
#define KPOP 500000000 //정수 매크로 상수
#define PI 3.14 //실수 매크로 상수
#define PRT printf("종료\n") //문자열 매크로 상수
```

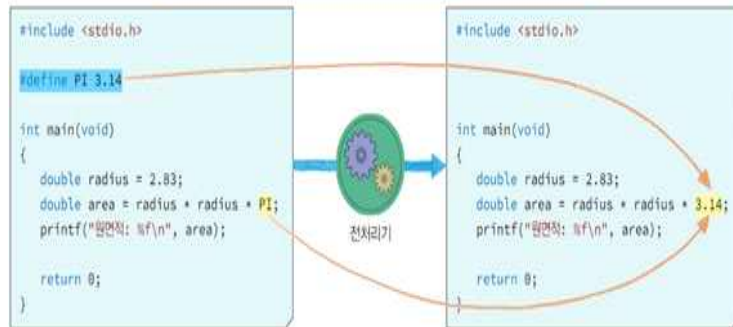


그림 4-5 매크로 상수와 전처리 과정

### 매크로 상수

전처리기(preprocessor)는 소스에서 정의된 매크로 상수를 모두 #define 지시자에서 정의된 문자열로 대체(replace) #define identifier\_name value #define에 정의된 identifier\_name은 전처리기에 의해 모두 value로 대체되어 컴파일한다. #define은 정수, 실수 또는 문자열 등의 상수를 KPOP, PI, PRT 등의 이름으로 정의, PI라는 매크로 상수. 전처리 과정에서 모두 3.14라는 실수로 값이 바뀐 소스로 컴파일하며 단 매크로 상수는 문자열 내부 또는 주석 부분에서는 대체되지 않는다.

다음은 #define에서 그 활용도를 높이기 위한 방안이 함수와 같이 인자(parameter)를 이용하는 방법이다. 하지만 주의점은 매크로를 구성하는 모든 인자와 외부에 괄호를 이용해야하며, 매크로 상수에서 매크로 이름과 시작괄호 사이에는 공백이 올 수 없다는 점이다.

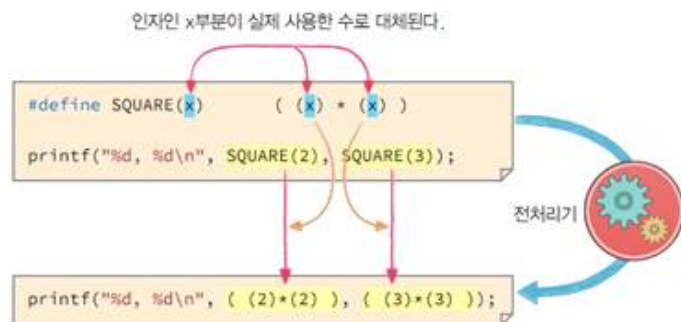


그림 4-7 인자가 있는 매크로의 치환

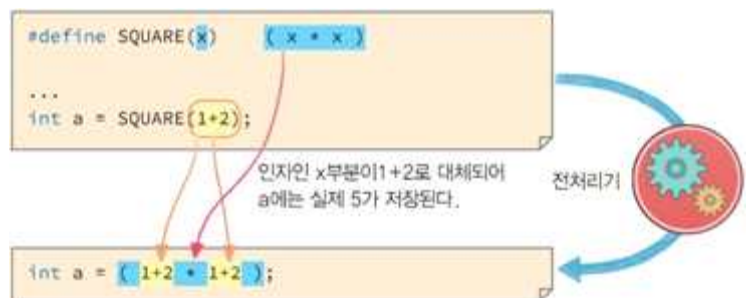
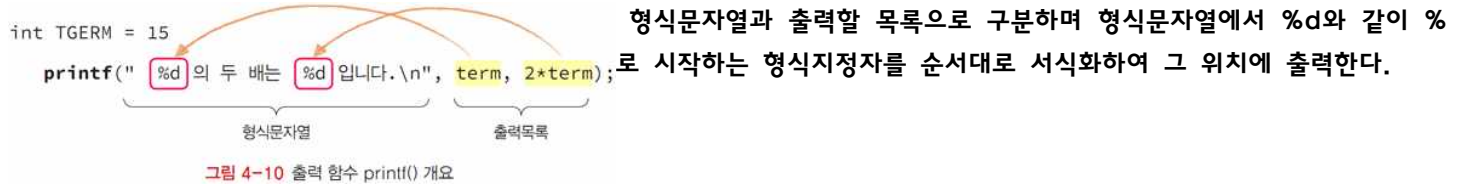


그림 4-8 매크로에서 괄호가 없는 인자의 잘못된 정의

## 4.2 출력함수 printf() .....

### printf()의 인자



함수 printf()의 첫 번째 인자인 형식문자열(format string)은 일반문자, 이스케이프 문자, 형식 지정자(format specification)로 구성되어 있으며 이스케이프 문자는 \와 \'같이 \로 시작하는 문자이다. %d와 %s와 같이 %로 시작하는 형식지정자가 존재한다.

`printf("%d대 연애에서 가장 중요한 것은 \"밀당\"이다.", 20);`

형식지정자 %d 위치에 바로 20이라는 정수가 출력  
 이스케이프 문자 \는 문자 "이 그대로 출력된다.

함수 printf()에서 정수 출력을 위한 형식 지정자는 정수의 십진수 출력을 위한 형식 지정자는 %d와 %i가 있다 또한 8진수로 출력하려면 %o를 이용해야 하며 앞 부분에 숫자 0이 붙는 출력을 하려면 %#o를 이용한다. 소문자의 십육진수로 출력하려면 %x와 대문자로 출력하려면 %X를 이용해야하며 16진수 앞에 0x또는 0X를 붙여 출력하려면 #을 삽입하여 %#x와 %#X를 이용해야 한다. 마지막으로 함수 printf()의 반환값은 출력한 문자의 수를 뜻한다.

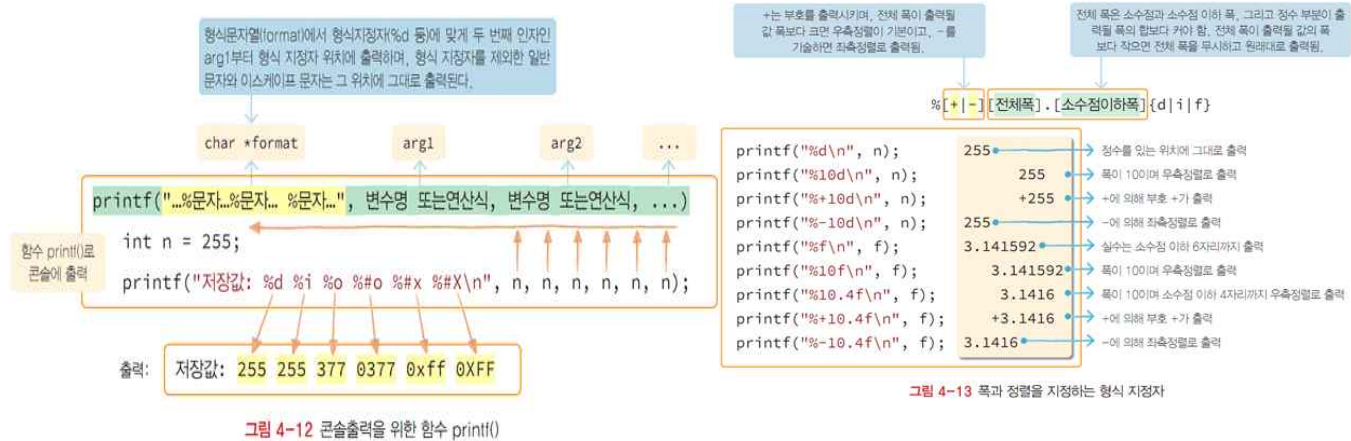


그림 4-12 콘솔출력을 위한 함수 printf()

double 실수의 출력을 위한 형식 지정자는 %lf이며 기본적으로 3.400000과 같이 소수점 6자리까지 출력한다. float 실수의 형식지정자는 %f이고 함수 printf()에서 실수 출력으로 %f, %lf 구분 없이 써도 되지만 구분할 필요가 있다. 출력 폭의 지정 시 출력 필드 폭이 출력 내용의 폭보다 넓으면 정렬은 기본이 오른쪽이며 필요하면 왼쪽으로 지정할 수 있다.

### 형식 문자의 종류 표

표 4-3 형식문자(type field characters) 종류

서식문자	자료형	출력 양식
c	char, int	문자 출력
d, i	int	부호 있는 정수 출력으로, lf는 long int, lld는 long long int형 출력
o	unsigned int	부호 없는 팔진수로 출력
x, X	unsigned int	부호 없는 십육진수 출력 x는 3ff와 같이 소문자 십육진수로, X는 3FF와 같이 대문자로 출력, 기본으로 앞에 0이나 0x, 0X는 표시되지 않으나 #이 앞에 나오면 출력
u	unsigned int	부호 없는 십진수(unsigned decimal integer)로 출력
e, E	double	기본으로 m.dddddexxx의 지수 형식 출력(정수 1자리와 소수점 이하 6자리, 지수승 3자리), 즉 123456.789이라면 1.234568e+005로 출력
f, lf	double	소수 형식 출력으로 m.123456 처럼 기본으로 소수점 6자리 출력되며, 정밀도에 의해 지정 가능, lf는 long double 출력
g, G	double	주어진 지수 형식의 실수를 e(E) 형식과 f 형식 중에서 짧은 형태(지수가 주어진 정밀도 이상이거나 -4보다 작으면 e나 E 사용하고, 아니면 f를 사용)로 출력, G를 사용하면 E가 대문자로
s	char *	문자열에서 '\0'가 나올 때 까지 출력되거나 정밀도에 의해 주어진 문자 수만큼 출력
p	void *	주소값을 십육진수 형태로 출력
%		%를 출력

### 옵션지정 문자의 종류 표

표 4-4 옵션지정 문자(flags) 종류

문자	기본(옵션)	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

## 4.3 입력함수 scanf()



그림 4-15 CT 촬영과 같이 표준입력의 자료를 스캔(scan)하여 주소가 지정된 변수에 저장

정자(format specification)는 %d, %c, %lf, %f와 같이 %로 시작한다. 또한 두 번째 인자부터는 키보드 입력 값이 복사되어 저장되는 입력변수 목록이므로 변수이름 앞에 반드시 주소연산자 &(ampersand)를 붙여야 한다.

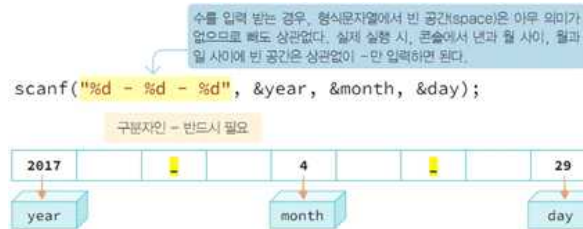


그림 4-17 입력값 사이에 특정 형식 지정

제어문자 %lf와 %f, %c에서 입력자료를 실수 double 형 변수에 저장할 때는 형식 지정자 %lf를 사용하며 입력자료를 실수 float형 변수에 저장할 때는 형식 지정자 %f를 사용한다. 입력 자료를 문자 char형 변수에 저장할 때는 제어문자 %c를 사용한다. 단, scanf\_s("%c", &ch, 1); 처럼 변수크기를 나타내는 인자가 하나 더 필요한데 여기서 ch는 char 변수를 말한다.

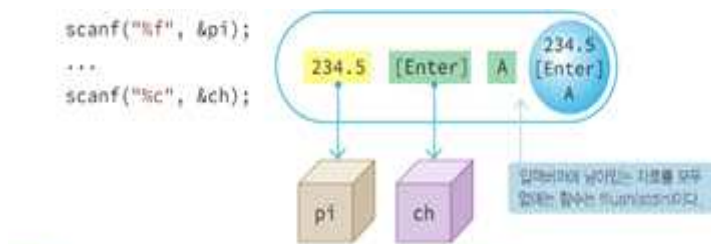


그림 4-19 문자 입력의 문제와 해결 방법

함수 scanf()에서 정수의 콘솔 입력값을 8진수로 인지하려면 %O를 사용한다. 마찬가지로 %x는 16진수로 인지한다. 다음은 함수 scanf()에서 이용되는 다양한 형식지정자 표이다.

표 4-8 함수 scanf()의 형식 지정자

형식 지정자	콘솔 입력값의 형태	입력 변수 인자 유형
%d	십진수로 인식	정수형 int 변수에 입력값 저장
%i	십진수로 인식하며, 단 입력값에 0이 앞에 붙으면 8진수로 0x가 붙으면 16진수로 인식하여 저장	정수형 int 변수에 입력값 저장
%u	unsigned int로 인식	정수형 unsigned int 변수에 입력값 저장
%o	8진수로 인식	정수형 int 변수에 입력값 저장
%x, %X	16진수로 인식	정수형 int 변수에 입력값 저장
%f	부동소수로 인식	부동소수형 float 변수에 입력값 저장
%lf	부동소수로 인식	부동소수형 double 변수에 입력값 저장
%e, %E	지수 형태의 부동소수로 인식	부동소수형 float 변수에 입력값 저장
%c	문자로 인식	문자형 char 변수에 입력값 저장
%s	일련의 문자인 문자열(string)로 인식	문자열을 저장할 배열에 입력값 저장
%p	주소(address) 값으로 인식	정수형 unsigned int 변수에 입력값 저장

함수 scanf()는 대표적인 입력함수이고 %d와 %lf 같은 동일한 형식 지정자를 사용한다. 'scan'이라는 단어는 스캐너와 같이 어떠한 자료를 훑어 복사하거나, 유심히 살펴본다는 의미로 첫 번째 인자는 형식문자열(format string)을 사용해야 하며 형식 지정자외의 문자열은 쓰지말 것을 권유한다. 형식지정자외의 문자열은 쓰지말 것을 권유한다.

지정된 형식지정자에 맞게 키보드로 적당한 값을 입력한 후 [Enter] 키를 누르기 전까지는 실행을 멈춰 사용자의 입력을 기다린다. 여러 입력값을 구분해주는 구분자(separator) -, /, 콤마(,) 등을 사용할 수 있지만, 입력된 구분자는 형식만 체크하고 저장하지 않는다. 만일 지정된 구분자와 입력 형식이 맞지 않으면 이후의 입력값은 제대로 저장되지 않으므로 주의가 필요하며 공백 이외에는 가급적 사용하지 않는게 좋다.

두 번의 scanf() 호출로, 콘솔에서 실수 234.5하나를 입력한 후 [Enter] 키를 누르고 다음 줄에 문자 'A'를 입력하여 변수 ch에 저장한다.

입력버퍼에는 [Enter] 키가 남아 있어, 두 번째 scanf()에서 char형 변수 ch에는 순서대로 [Enter]인 문자 '\n'가 저장되고, 실제 문자 'A'는 저장되지 않는 문제가 발생하는데 이러한 문제를 해결하는 방법은 두 가지이다.

첫 번째는 버퍼에 남아있는 [Enter] 키를 함수 fflush(stdin)를 호출하여 없애버리는 방법이며 두 번째는 문자를 입력 받는 형식지정자 %c 앞에 "공백문자를 넣어" 형식문자열을 " %c"로 지정하여 아직 입력버퍼에 남아있는 [Enter] 키가 %c 앞에 공백문자로 인식되어 무시되고, 이어 커서 위치에 입력되는 'A'가 변수 ch에 저장되는 방법이다.

문자의 입출력 함수 getchar()는 영문 'get character'의 의미로 문자 하나를 입력하는 매크로 함수이며 putchar()는 'put character'로 반대로 출력하기 위한 매크로 함수이다. 이 함수를 이용하려면 헤더파일 stdio.h 가 필요하며 getchar()는 인자 없이 함수를 호출한다. 입력된 문자값을 자료형 char나 정수형으로 선언된 변수에 저장한다. putchar('a')는 출력할 문자를 인자로 호출하는데 인자인 'a'를 출력하는 함수로 사용한다.



## 5.1 연산식과 다양한 연산자

연산자와 피연산자, 연산식과 연산값에서 연산식은

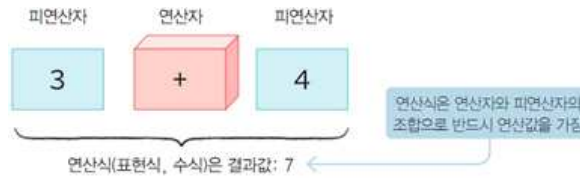


그림 5-1 연산식과 연산식 값(연산식 평가 또는 결과값)

일상 생활에서 사용하는  $(3 + 4 * 5)$ 와 같은 간단한 식을 말하며 변수와 다양한 리터럴 상수, 함수의 호출 등으로 구성되는 식이다. 연산자(operator)는 산술연산자  $+$ ,  $-$ ,  $*$  기호와 같이 이미 정의된 연산을 수행하는 문자 또는 문자조합 기호이며 피연산자(operand)는 연산(operation)에 참여하는 변수나 상수를 말한다. 연산식  $3 + 4$ 에서 ' $+$ '는 연산자이고, 3과 4는 피연산자이고 7처럼 항상 하나의 결과 값을 가진다.

단항연산자



이항연산자

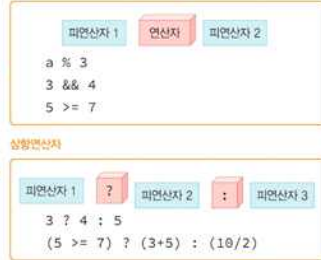


그림 5-3 단항, 이항, 삼항연산자

연산자는 다양한 종류의 연산자가 있는데 단(일)항(unary), 이항(binary), 삼항(ternary) 연산자로 나눌 수 있다. 연산자는 연산에 참여하는 피연산자(operand)의 갯수에 따라 구분하며 부호를 표시하는  $+$ ,  $-$ 와 증감연산자  $++$ ,  $--$ 는 단항 연산자

덧셈, 뺄셈의  $+$ ,  $-$ ,  $*$ ,  $/$  등의 연산은 이항연산자이며 삼항연산자는 조건연산자 ' $?$ ' ' $:$ '가 유일하다. 또한 단항연산자는 연산

자의 위치에 따라 전위와 후위로 나뉘는데  $++a$ 처럼 연산자가 앞에 있으면 전위(prefix) 연산자,  $a++$ 와 같이 연산자가 뒤에 있으면 후위(postfix) 연산자라고 부른다.

표 5-1 다양한 산술 연산식

연산식	설명	연산값	연산식	설명	연산값
$-a$	부호연산자	10	$-b + 2.5$	부호연산자	0.0
$2 - a$	빼기연산자	-3	$a / b + b * 2$	$(a / b) + (b * 2)$	5.0
$2 * a + 3$	$(2 * a) + 3$	13	$a * 2 / b - a$	$((a * 2) / b) - a$	-1.0
$10 - a / 2$	$10 - (a / 2)$	8	$a + b * 2 / a$	$a + ((b * 2) / a)$	6.0
$10 \% a + 10 / a$	$(10 \% a) + (10 / a)$	2	$a \% b$	실수는 %에 오류	오류

산술연산자는  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $\%$ 가 있는데 먼저  $\%$  나머지(remainder, modulus) 연산자는 피연산자로 정수만 가능하며 나누기 연산식  $10 / 4$ 는 연산값이 2이지만 나머지 연산식  $a \% b$ 의 결과는  $a$ 를  $b$ 로 나눈 나머지 값이다.  $\%$ 의 피연산자는 반드시 정수가 되어야 하며 실수이면 오류가 발생한다. 부호 연산자(단항 연산자)는 변수의 부호로 표기하는 연산자이다.

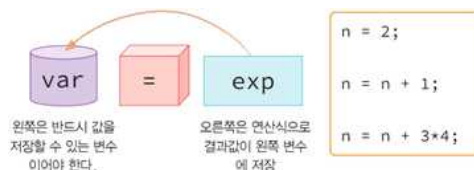


그림 5-7 대입연산자 수행 방법

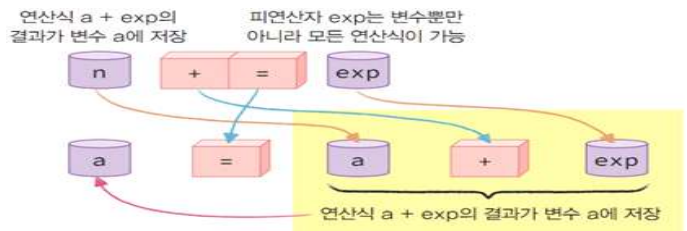


그림 5-9 축약 대입연산자의 연산 방법

```
int n = 10;
printf("%d\n", n++);
printf("%d\n", n);
```

출력: 10, 11

```
int n = 10;
printf("%d\n", ++n);
printf("%d\n", n);
```

출력: 11, 11

```
int n = 10;
printf("%d\n", n--);
printf("%d\n", n);
```

출력: 10, 9

```
int n = 10;
printf("%d\n", --n);
printf("%d\n", n);
```

출력: 9, 9

잘못된 사용 예

```
int a = 10;
++300; //상수에는 증감연산자를 사용할 수 없다.
(a+1)--; //일반 수식에는 증감연산자를 사용할 수 없다.

//8나りの 연산식에 동일한 변수의 증감연산자는 사용하지 말자.
a = ++a * a--;
```

그림 5-12 증감연산자 사용

대입연산자(assignment operator) = 연산자 오른쪽의 연산값을 변수에 저장하는 연산자이다. 대입연산식  $a = a + b$

중복된  $a$ 를 생략하고 간결하게  $a += b$  라고 표기하는 방법을  $(-=, *=, /=, \%=)$ 을 축약 대입연산자라고 하며 산술연산자와 대입연산자를 이어 붙인 연산자  $+=$  즉  $a += 2$ 는  $a = a + 2$ 의 대입연산을 의미한다. 연산자  $++$ ,  $--$ 에서 증가연산자  $++$ 는 변수값을 각각 1 증가시킨다는 의미이며 감소연산자  $--$ 는 1 감소라는 의미를 갖고 있다.  $n++$ 와  $++n$  모두  $n = n + 1$ 의 기능을 수행하고  $n--$ 와  $--n$  도 동일하게  $n = n - 1$ 의 기능을 수행한다. 하지만 연산의 일부나 함수의 인자로 사용되면 전위/후위 결과는 다르다.  $n++$ : 후위(postfix)같은 경우에는 1 증가되기 전 값이 연산에 사용되지만  $++n$ : 전위(prefix) 같은 경우에는 1 증가된 값이 연산에 사용된다.

5.2 관계와 논리, 조건과 비트연산자 .....

두 피연산자의 크기를 비교하기 위한 연산자를 관계연산자라고 하며 비교 결과가 참이면 0, 거짓이면 1이다. 관계연산자 !=, >=, <=는 연산 기호의 순서 주의해야 하며 관계연산자 ==는 대입연산자 =와 혼동하지 않도록 주의해야 한다. 정수형, 실수

표 5-2 관계연산자의 종류와 사용

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

형, 문자형 등이 피연산자가 될 수 있지만 피연산자가 문자인 경우 문자 'a'는 코드값이 97이고 문자 'Z'는 코드값이 90이므로 연산식 ('Z' < 'a')는 1인 참을 의미한다. 즉 문자 'O' < '1' < '2' < '3' < ... < '9'인 관계가 있으며 'a' < 'b' < 'c' < ... < 'x' < 'y' < 'z' 관계가 있고, 대문자도 마찬가지이며, 'Z' < 'a'로 소문자는 대문자보다 모두 크다.

C언어에선 세 가지의 논리연산자 &&, ||, !을 제공하는데 논리연산자 &&, ||, !은 각각 and, or, not 의미한다. 결과가 참이면 1 거짓이면 0을 반환하며 C 언어에서 참과 거짓의 논리형은 따로 없다. 0, 0.0, \0은 거짓을 의미하지만 0이 아닌 모든 정수와 실수, 그리고 널(null) 문자 '\0'가 아닌 모든 문자와 문자열은 모두 참을 의미한다. 논리연산자 &&는 두 피연산자가 모두 참(0이 아니어야)이면 결과가 1(참)이며 나머지 경우는 모두 0으로 처리한다. 논리연산자 ||는 두 피연산자 중에서 하나만 참(0이 아니어야)이면 1이고, 모두 0(거짓)이면 0으로 처리한다. 논리연산자 !는 단항연산자로 피연산자가 0이면 결과는 1이고 참(0이 아닌 값)이면 결과는 0이다. 비트논리연산자 : &, |는 각 비트별로 논리값을 계산하는 연산자이다.

x	y	x & y	x    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 && 31

!2 && 'a'0

3>4 && 4>=20

1 || '\0'1

2>=1 || 3 <=01

0.0 || 2-20

!01

그림 5-13 논리연산자의 연산 결과

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

연산자 ?: 조건연산자는 조건에 따라 주어진 피연산자가 결과값이 되는 삼항연산자라고 한다. 즉 연산식 (x ? a : b)에서 피연산자는 x, a, b 세 개다. 피연산자인 x가 참이면(0이 아니면) 결과는 a이며, x가 0이면(거짓) 결과는 b로 처리된다. 비트 논리연산자는 피연산자 정수값을 비트 단위로 논리 연산을 수행하는 연산자로 비트 연산은 각 피연산자를 int형으로 변환하여 연산하며 결과도 int 형이다. 다음은 각 비트의 연산 방법이다.

x(비트 1)	y(비트 2)	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

어릴 적 학교에서 오른쪽 또는 왼쪽으로 줄줄이 자리 이동을 한 경험이 있을 것이다. 바로 비트 이동 연산자(bit shift operators) >>, << 는 연산자의 방향인 왼쪽이나 오른쪽으로, 비트 단위로 줄줄이 이동시키는 연산자이다.

5.3 형변환 연산자와 연산자 우선순위 .....

올림변환은 작은 범주의 자료형(int)에서 보다 큰 범주인 형(double)으로의 형변환을 하는 것이다. 올림변환은 형 넓히기라고도 부르며 올림변환은 정보의 손실이 없으므로 컴파일러에 의해 자동으로 수행이 가능하다. 컴파일러가 자동으로 수행하는 형 변환은 묵시적 형변환이라고 한다. 내림변환은 큰 범주의 자료형(double)에서 보다 작은 범주인 형(int)으로의 형변환이며 대입연산 int a = 3.4에서 내림변환이 필요하다. 컴파일러가 스스로 시행하는 묵시적 내림변환의 경우 정보의 손실이 일어날 수 있으므로 경고를 발생하여 알려준다. 그러므로 프로그래머의 명시적 형변환이 필요하다.



그림 5-21 피연산자의 자동 올림변환



그림 5-22 대입연산에서의 내림변환과 올림변환

명시적 형변환(explicit type conversion)은 형변환 연산자를 사용하여 자료형을 지정하며 ‘(자료형) 피연산자’ 형식으로 하여 지정한 자료형으로 변환한다.상수나 변수의 정수값을 실수로 변환하려면 올림변환을 사용하며 (double) 7의 결과는 7.0이다. 실수의 소수부분을 없애고 정수로 사용하려면 내림변환을 사용해야하는데 (int) 3.8의 결과는 3과 같이 단항연산자인 형변환 연산자는 모든 이항연산자보다 먼저 계산한다. 다음은 연산자의 우선순위이다.

표 5-9 C 언어의 연산자 우선순위

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)
	++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	,	콤마	콤마	-> (좌에서 우로)



## 6.1 제어문 개요

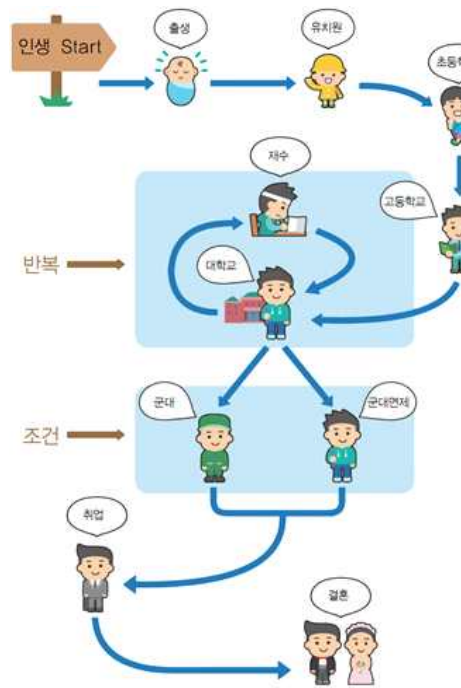


그림 6-1 인생여정과 프로그램의 실행 흐름

제어문의 종류에서 순차적(sequential) 실행은 지금까지 배워 온 프로그램의 실행 순서의 원칙이며 main 함수 내부에서 배치된 문장이 순차적으로 실행되는 흐름이다. 비순차적 실행의 제어문에서 순차적 실행만으로 프로그램을 모두 작성한다면 매우 비효율적이며 프로그램의 실행 순서를 제어하는 제어문(control statement)을 제공하여 선택과 반복 등 순차적인 실행을 변형한다.

다음은 제어문의 종류이다.

조건선택 구문이란 두 개 또는 여러 개 중에서 한 개를 선택하도록 지원하는 구문

### 조건선택

#### 조건에 대한 선택 구문

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

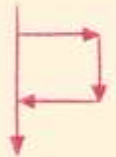


그림 6-2 조건선택

반복(순환)구문이란 정해진 횟수 또는 조건을 만족하면 정해진 몇 개의 문장을 여러 번 실행하는 구문이다.

분기 구문은 작업을 수행도중 조건에 따라 반복이나 선택을 하여 빠져나가거나 일정 구문을 실행하지 않고 다음 반복을 실행하거나 지정된 위치로 이동하거나 작업 수행을 마치고 이전 위치로 돌아가는 구문이 있다.

### break 문

작업을 수행 도중 조건에 따라 반복이나 선택을 빠져 나가기

### continue 문

일정구문을 실행하지 않고 다음 반복을 실행

### goto 문

지정된 위치로 이동

### return 문

작업 수행을 마치고 이전 위치로 돌아가는 구문

### 반복 순환

#### 반복조건에 따라 일정영역의 반복 구문

- for
- while
- do while



그림 6-3 반복순환

### 분기처리

#### 지정된 영역으로 실행을 이동하는 구문

- break
- continue
- goto
- return



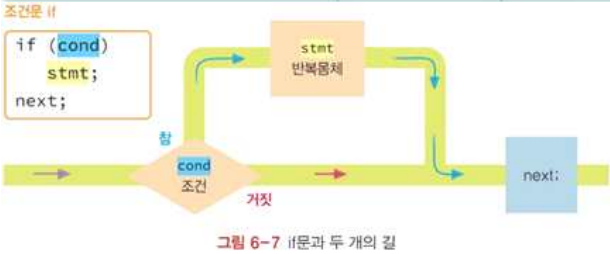
그림 6-4 분기처리

6.2 조건에 따른 선택 if 문 .....

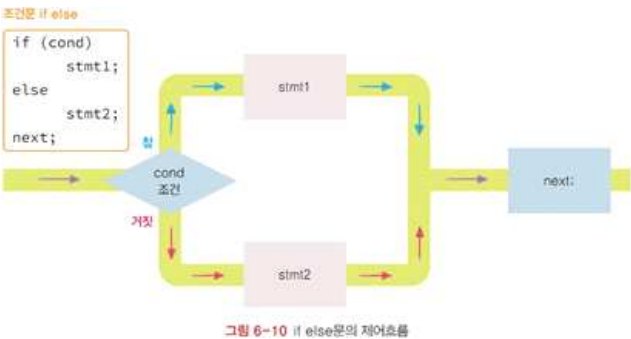
표 6-1 조건에 따라 선택이 발생하는 일상생활에서의 사례

조건 선택의 예	기준변수	조건 표현의 의사코드
온도가 32도 이상이면 폭염 주의를 출력	온도 temperature	만일 (temperature >= 32) printf("폭염 주의");
낮은 혈압이 100이상이면 '고혈압 초기'로 진단	혈압 low_pressure	만일 (low_pressure >= 100) printf("고혈압 초기");
속도가 40km와 60km 사이이면 "적정 속도"라고 출력	속도 speed	만일 (40 <= speed && speed <= 60) printf("적정 속도");
운전면허 필기시험에서 60점 이상이면 합격, 아니면 불합격 출력	시험 성적 point	만일 (point >= 60) printf("면허시험 합격"); 아니면 printf("면허시험 불합격");
남성이면 체력 테스트에서 80이상이면 합격이고, 아니면 불합격, 여성이면 70이상이면 합격, 아니면 불합격	여성, 남성 type  체력 점수 point	만일 남성이면 (type == 1) 만일 (point >= 80) printf("남성: 합격"); 아니면 printf("남성: 불합격"); 아니고 만일 여성이면 (type == 2) 만일 (point >= 70) printf("여성: 합격"); 아니면 printf("여성: 불합격");

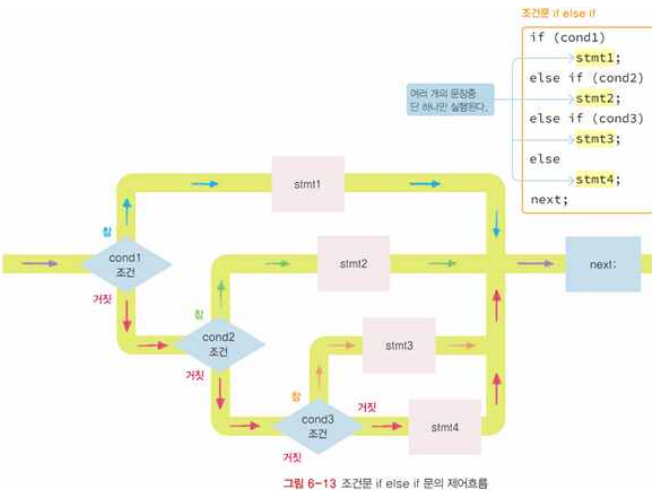
다음 표는 일상생활에서 조건에 따라 선택이 발생하는 다양한 예를 지니고 있다. 첫 번째 예에서 조건의 기준은 온도이며 이 온도를 temperature에 저장한다면 이 변수가 조건의 기준변수가 되고, 관계식으로 조건식을 표현할 수 있으며 다음과 같은 의사코드로 조건 선택의 예를 표현할 수 있다.



이 그림은 가장 간단한 if문의 형태이다. 만약 조건을 만났을 경우 그 조건이 참이면 stmt를 실행하며 거짓일 시 실행하지 않는다.



이 그림은 조건 만족 여부에 대한 선택 if else이다. 조건 cond를 만족하면 stmt1을 실행하지만 조건 cond를 만족하지 않으면 stmt2를 실행하는 stmt1과 stmt2 둘 중의 하나를 선택하는 구문이다.



이 그림은 else 이후에 if else를 필요한 횟수만큼 반복하여 사용하는 문장이다. 하나씩 조건을 체크하며 참일 경우 거짓일 경우를 처리하는 문장이다. 하지만 stmt1에서 stmt4에 이르는 여러 문장 중에서 실행되는 문장은 단 하나라는 것을 기억해야한다.

else는 문법적으로 같은 블록 내에서 else가 없는 가장 근접한 상위의 if문에 소속된 else로 해석된다. 따라서 else의 혼란을 방지하려면 블록을 이용할 것을 권유한다.

## 6.3 다양한 선택 switch 문 .....

연산식의 정수 또는 문자 선택에서 다중선택 구문인 switch문을 사용할 때 조건식이 정수 등호식이고, if else가 여러 번 계속 반복되는 구문을 좀 더 간략하게 구현하면 주어진 연산식이 문자형 또는 정수형으로 그 값에 따라 case의 상수값과 일치하는 부분의 문장들을 수행하는 선택 구문이다.

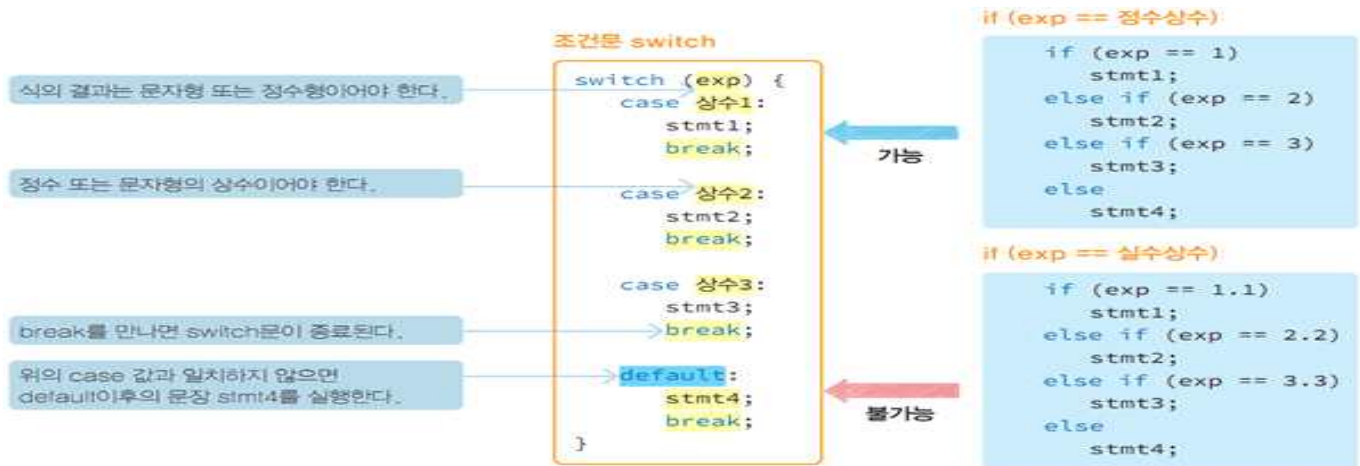


그림 6-19 switch 문 문장구조

다음 예제는 1에서 4까지 정수 중에서 선택한 번호에 따라 표준 입력한 두 실수의 더하기, 빼기, 곱하기 나누기를 실행하는 프로그램이다.

```

// file: switch.c
#define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의

#include <stdio.h>

int main(void)
{
    double x, y, result;
    int op;

    printf("두 실수 입력: ");
    scanf("%lf %lf", &x, &y);
    printf("연산종류 번호선택 1(+), 2(-), 3(*), 4(/): ");
    scanf("%d", &op);

    switch (op) {
    case 1:
        printf("%.2f + %.2f = %.2f\n", x, y, x + y);
        break;
    case 2:
        printf("%.2f - %.2f = %.2f\n", x, y, x - y);
        break;
    case 3:
        printf("%.2f * %.2f = %.2f\n", x, y, x * y);
        break;
    case 4:
        printf("%.2f / %.2f = %.2f\n", x, y, x / y);
        break;

    default:
        printf("번호를 잘못 선택했습니다.\n");
        break; //생략가능
    }

    return 0;
}
    
```

Microsoft Visual Studio 디버그 콘솔

```

두 실수 입력: 3.765 6.987
연산종류 번호선택 1(+), 2(-), 3(*), 4(/): 1
3.77 + 6.99 = 10.75

C:\Users\김범기\source\repos\Project6\Debug\Project6.exe
이 창을 닫으려면 아무 키나 누르세요...
    
```

case 문 내부에 break 문이 없다면  
일치하는 case 문을 실행하고  
break 문을 만나기 전까지 다음 case 내부 문장을 실행  
default는 일치하는 case가 없을 때 실행된다.



## 7.1 반복 개요와 while 문 .....

반복(repetition)이란 같거나 비슷한 일을 여러 번 수행하는 작업이며 순환(loop, 루프)은 반복과 같은 의미이다.

롤러코스터의 원형 궤도처럼 원래 고리 또는 순환이라는 의미가 루프(loop)라고 할 수 있다. 반복문의 종류에는 3가지가 있다.

반복문 while은 조건식이 참이면(0이 아니면) 반복을 더 실행하며 조건식이 반복문체 앞에 위치한다.

do while문은 무조건 한 번 실행 한 후 조건을 검사하고 이때 조건식이 참이면 반복을 더 실행한다.

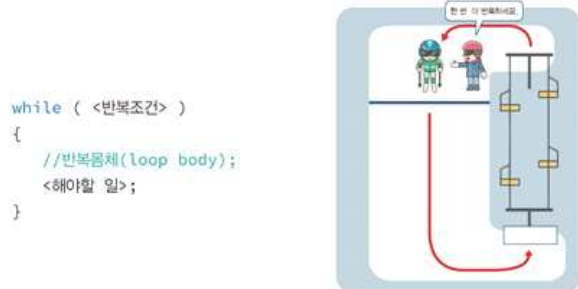


그림 7-2 while 반복

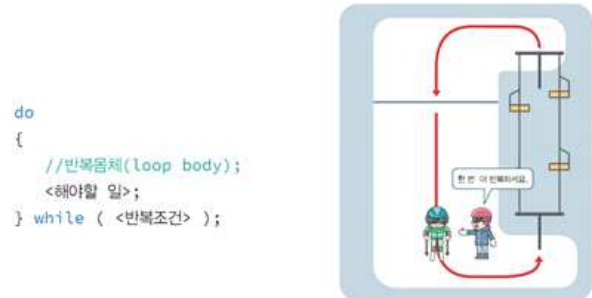


그림 7-3 do while 반복



그림 7-4 for 반복

반복문 for는 숫자로 반복 횟수를 제어하는 반복문이며 명시적으로 반복 횟수를 결정할 때 주로 사용된다.

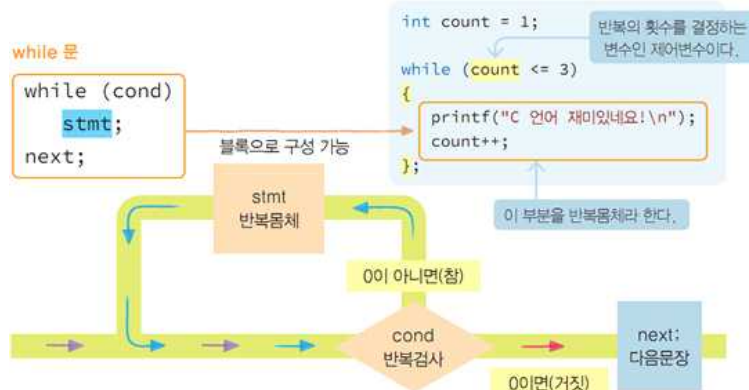


그림 7-9 반복 while문과 제어흐름

문장 while (cond) stmt;에서 cond를 평가하여 0이 아니면(참이면) 반복문체인 stmt를 실행하고 반복은 cond가 0(거짓)이 될 때까지 계속된다. 반복이 실행되는 stmt를 반복문체(repetition body)라 부르며 필요하면 블록으로 구성할 수 있고 while 문은 for나 do while 반복문보다 간단하다는 장점이 있다.

아래 그림을 보면 후위 증가연산자 n++의 연산값은 증가되기 이전 값이므로 반복이 시작된 1부터 5까지 출력한다.

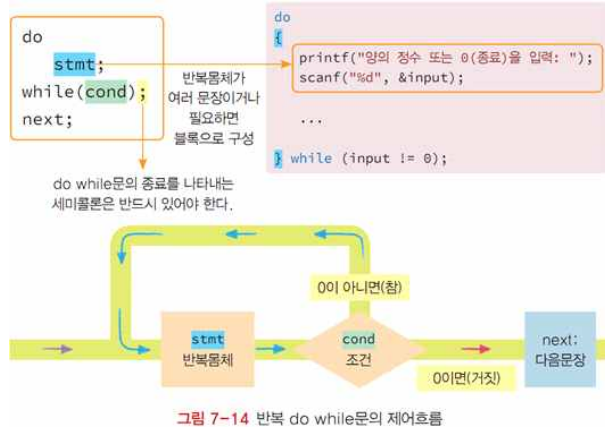
연산식 n++에서 5가 출력되면, n이 6이 되고, while 조건식 (6 <= 5) 값이 0이 되어 while 문장을 빠져 나오는 것을 볼 수 있다.

```
int n = 1;
//정수값을 1씩 증가시키면서 출력 반복
printf("%d\n", n++);
printf("%d\n", n++);
printf("%d\n", n++);
printf("%d\n", n++);
printf("%d\n", n++);
```

```
#define MAX 5
...
int n = 1;
while (n <= MAX)
    printf("%d\n", n++);
```

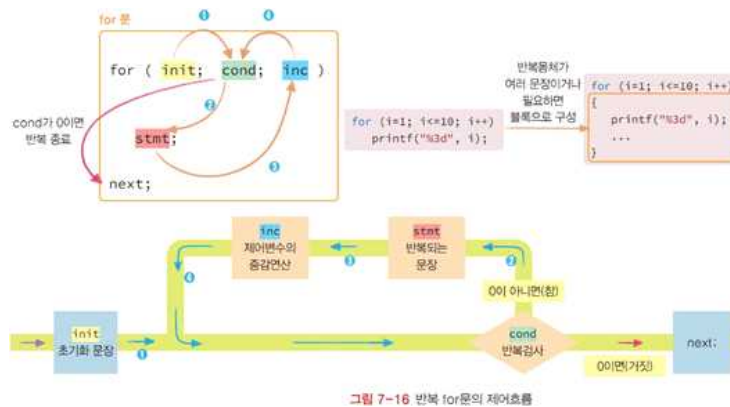
그림 7-13 1에서 5까지 출력하는 while

## 7.2 do while 문과 for 문



문장 `do stmt; while (cond);`에서 가장 먼저 `stmt`를 실행한 이후 반복조건인 `cond`를 평가하며 0이 아니면(참이면) 다시 반복문체인 `stmt`를 실행한다. 0이면(거짓이면) `do while` 문을 종료한다. 하지만 `while` 이후의 세미콜론은 반드시 필요 하므로 주의를 해야한다.

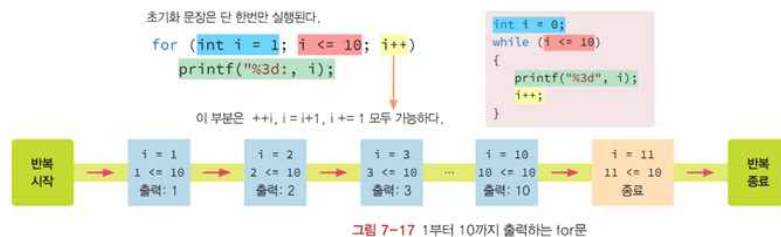
반복문 `for (init; cond; inc) stmt;`에서 각각 초기화(initialization); 반복조건 검사; 제어변수의 증감(increment)을 수행한다. 하지만 주의해야 할 점이 있는데 `for( ; ; )`의 괄호 내부에서 세미콜론으로 구분되는 항목은 모두 생략 가능하지만 2개의 세미콜론은 반드시 필요하다. 또 반복조건 `cond`를 아예 제거하면 반복은 무한히 계속되어 무한 루프에 빠질 수 있으며 반복할 문장인 반복문체 `stmt`가 여러 개라면 반드시 블록으로 구성해야 한다.



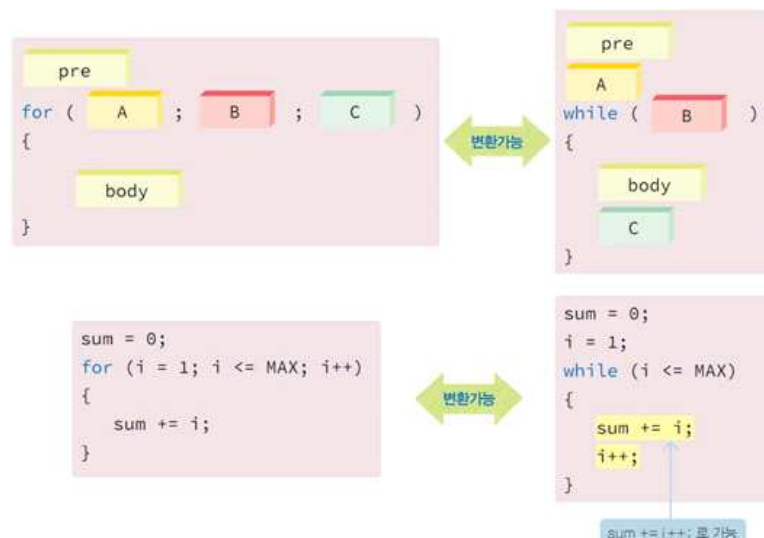
`for (i=1; i<=10; i++) printf("%3d",i);`

1부터 10까지 출력

- 1 초기화를 위한 `init`를 실행한다. 이 `init`는 단 한번만 수행된다.
- 2 반복조건 검사 `cond`를 평가  
참이면 반복문의 몸체에 해당하는 문장 `stmt`를 실행  
거짓이면 `for` 문을 종료
- 3 반복문체인 `stmt`를 실행한 후 증감연산 `inc`를 실행한다.
- 4 다시 반복조건인 `cond`를 검사하여 반복한다.

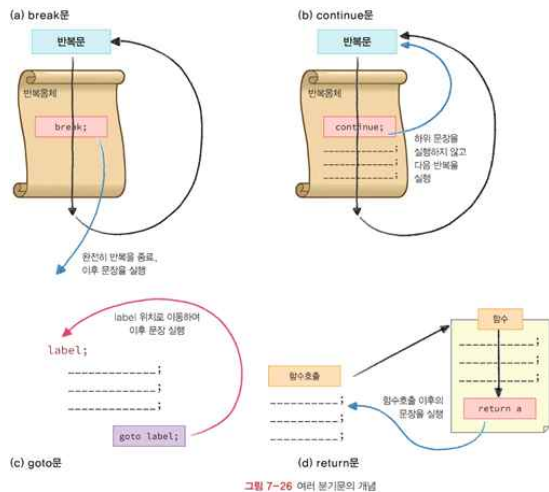


다음 그림은 1에서 10까지 출력하는 프로그램이다. 반복횟수를 제어하는 제어변수 `i`를 1로 초기화한 뒤 조건검사 `i <= 10`를 이용하여 변수 `i`를 출력하고 변수 `i`와 같이 반복의 횟수를 제어하는 변수를 제어변수라고 한다.



`for` 문은 주로 반복횟수를 제어하는 제어변수를 사용하여 초기화와 증감부분이 있는 반복문에 적합하다. `while` 문은 문장구조가 간단하므로 다양한 구문에 이용되고, 특히 `while` 문은 반복횟수가 정해지지 않고 특정한 조건에 따라 반복을 결정하는 구문에 적합하다. 또한 `for` 문과 `while` 문은 서로 변환이 가능한데 옆 그림 과 같이 먼저 `for` 문에서 반복을 위해 필요한 인자들을 `while` 문으로 수정할 경우에는 변수 선언 부분을 위에서 선언해주며 조건 부분을 `while` 문의 인자로 넣고 똑같이 몸체에서 수행할 부분을 넣으며 마지막으로 반복의 종료를 위해 증가되는 부분을 마지막에 넣어 준다.

## 7.3 분기문



C언어 분기문(return, break, continue)이란 프로그램의 순차적 수행 순서에 따르지 않고 다른 명령을 수행하도록 이행 시키는 명령이다. 분기문에는 여러 종류가 있고, 각각 다른 특징을 가집니다. 분기문은 정해진 부분으로 바로 실행을 이동(jump)하는 기능을 수행하며 C가 지원하는 분기문의 종류는 break, continue, goto, return 문이 있다.

반복내부에서 반복을 종료하려면 break문장을 사용한다. 만일 반복문이 중첩되어 있다면 break를 포함하는 가장 근접한 내부반복을 종료한다. continue 문이 위치한 이후의 반복문체의 나머지 부분을 실행하지 않고 다음 반복을 계속 유지하는 문장이며 continue 이후의 문장은 실행되지 않고 뛰어 넘어간다. 반복문 while과 do while 문에서 continue를 만나면 조건 검사로 이동하여 실행되며 반복문 for 문에서는 continue 문을 만나면 증감 부분으로 이동하여 다음 반복을 실행한다.

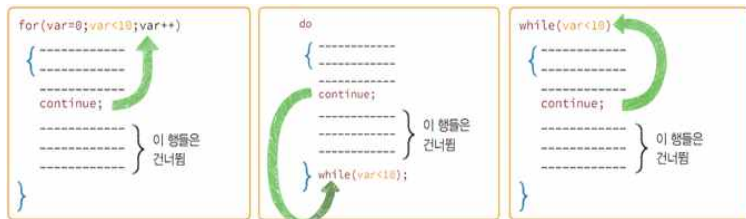


그림 7-29 다양한 continue문 이후의 실행 순서

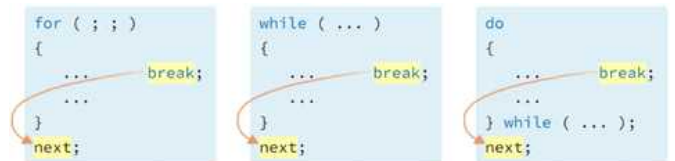


그림 7-27 반복문의 break

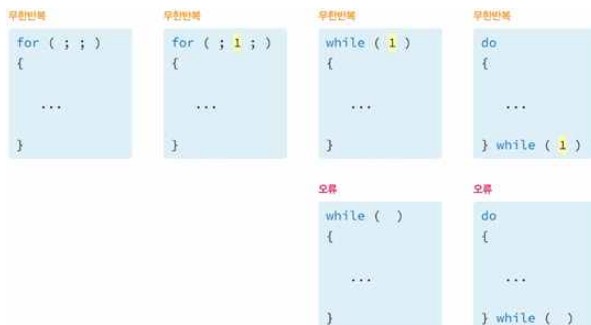


그림 7-31 반복문의 무한반복

다음예제는 무한 반복을 하는 예제인데 무한 반복이란 반복문에서 무한히 반복이 계속되는 것을 말하며 while과 do while은 반복조건이 아예 없으면 오류가 발생한다.

```
// file: break.c
#define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의

#include <stdio.h>

int main(void)
{
    int input;

    do {
        printf("\t\t [1] 한식\n");
        printf("\t\t [2] 양식\n");
        printf("\t\t [3] 분식\n");
        printf("\t\t [4] 기타\n");
        printf("메뉴 번호 선택 후 [Enter] : ");
        scanf("%d", &input);
        printf("선택 메뉴 %d\n", input);

        if (input <= 4 && input >= 1)
            break;
    } while (1);

    return 0;
}
```



다음 예제는 원하는 메뉴를 선택하는 예제이며 간단한 음식 메뉴 구성으로 사용자가 메뉴를 선택하면 프로그램을 종료하지만 적당한 메뉴를 선택하지 못하면 선택할 때까지 반복을 실행한다.

## 7.4 중첩된 반복문

이 예제 코드를 보면 제일 처음  $m$ 을 증감 연산자를 사용하여 반복문을 돌리며 그 안에 다시  $n$ 을 증감 연산자를 사용하여 반복문을 다시 돌리는 것을 확인 할 수 있다. 이렇게 반복문 안에 또 다른 반복문이 들어가있는 형식을 중첩된 반복문이라고 하는데 이 코드를 실행하여 보면 외부반복에서 1에서 5까지, 내부반복에서 1에서 7까지 반복하면서 각각의 변수값을 출력하는 것을 확인 할 수 있다.

```

C:\Users\김범기\source\repos\Project6\Debug\Project6.exe
이 창을 닫으려면 아무 키나 누르세요...

```

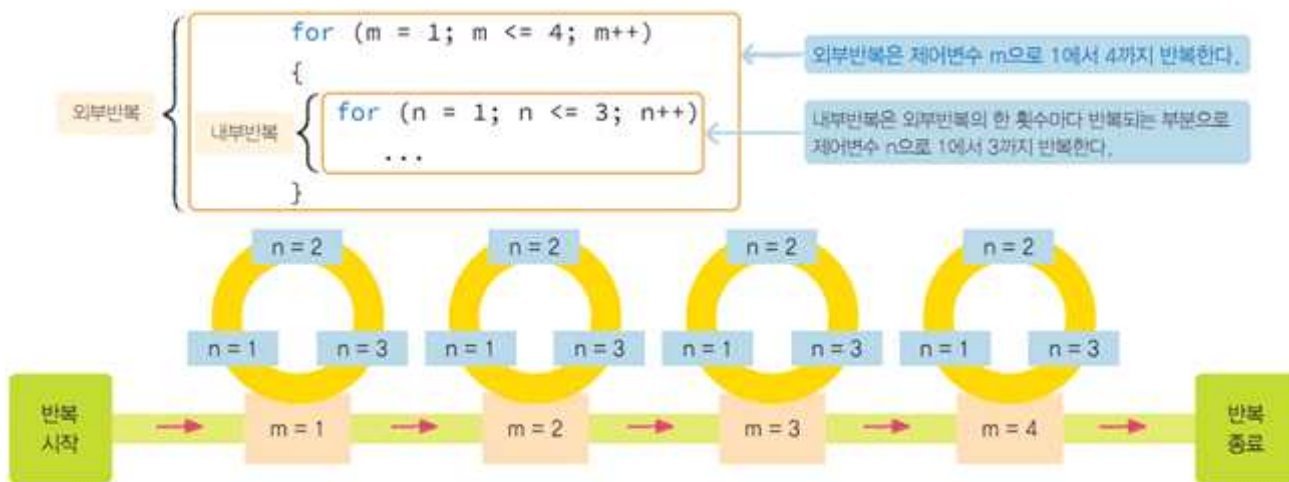


그림 7-33 중첩된 for문에서 제어변수의 변화(외부반복 제어변수: m, 내부반복 제어변수: n)

그럼 중첩된 반복문을 사용하여 구구단을 출력하는 예제를 만들어 보자.

```

// file: mtable.c
#include <stdio.h>
#define MAX 9

int main(void)
{
    printf("=== 구구단 출력 ===\n");
    for (int i = 2; i <= MAX; i++)
    {
        printf("%d단 출력\n", i);
        for (int j = 2; j <= MAX; j++)
        {
            printf("%d * %d = %d ", i, j, i * j);
            printf("\n");
        }
    }

    return 0;
}

```

```

C:\Users\김범기\source\repos\Project6\Debug\Project6.exe (프로세스 23004개)
이 창을 닫으려면 아무 키나 누르세요...

```



## 8.1 포인터 변수와 선언

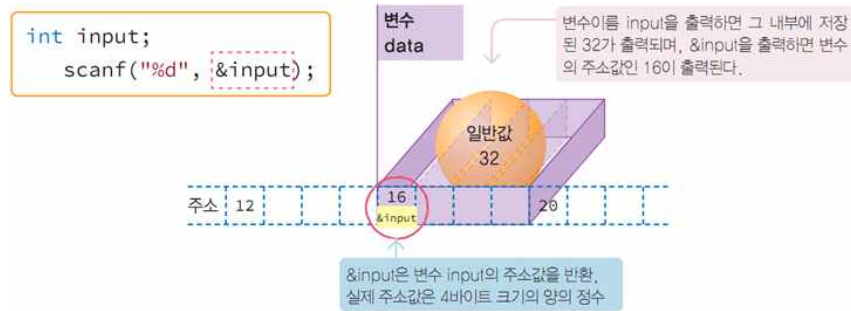


그림 8-2 주소연산자 & 이해

주소(address)는 메모리 공간은 8비트인 1 바이트마다 순차적인 고유한 번호이다. 메모리 주소는 저장 장소인 변수이름과 함께 기억 장소를 참조하는 또 다른 방법이다. 함수 scanf\_s()를 사용하면서 인자를 '&변수이름'으로 사용할 때 바로 &(ampersand)가 피연산자인 변수의 메모리 주소를 반환하는 주소연산자이며 함수 scanf()에서 입력값을 저장하는 변수의 주소값이 인자의 자료형이다.

포인터 변수는 주소값을 저장하는 변수이며 변수의 주소값은 반드시 포인터 변수에 저장된다. 일반 변수에는 일반 자료 값이 저장되며 포인터 변수는 일반 변수와 구별되며 선언방법이 다르다.

선언 방법은 먼저 포인터 변수 선언에서 자료형과 포인터 변수 이름 사이에 연산자 \*(asterisk)를 삽입한 뒤 다음의 printf, ptrshort, ptrchar, ptrdouble은 모두 포인터 변수이다. 예를 들면 int \*pprintf 라고 선언할 시 int 포인터 pprintf 라고 읽으며 변수 자료형이 다르면 그 변수의 주소를 저장하는 포인터의 자료형도 반드시 달라야 한다.

포인터 변수선언

자료형	*변수이름 ;
int *pprint;	int *pprint; //가장 선호
short *ptrshort;	short *ptrshort;
char *ptrchar;	char * ptrchar;
double *ptrdouble;	double *ptrdouble;

그림 8-4 포인터 변수선언 구분

int data = 100;	변수 data는 정수 int를 저장하는 일반변수
int *pprint;	변수 pprint는 정수 int를 저장하는 일반변수의 주소를 저장하는 포인터 변수
pprint = &data;	포인터 pprint는 이제 'data'를 가리킨다 (data 주소값을 갖는다).

그림 8-5 포인터 변수와 일반 변수의 주소 저장

포인터 변수도 선언된 후 초기값이 없으면 의미 없는 쓰레기(garbage) 값이 저장된다. 문장 pprint = &data; 아래는 포인터 변수 pprint에 변수 data의 주소를 저장하는 예제코드이다.

```
//file: pointer.c
#include <stdio.h>

int main(void) {
    int data = 100;
    int* pprint;
    pprint = &data;

    printf("변수명      주소값      저장값\n");
    printf("-----\n");
    printf("data      %p  %8d\n", &data, data);
    printf("pprint    %p  %p\n", &pprint, pprint);

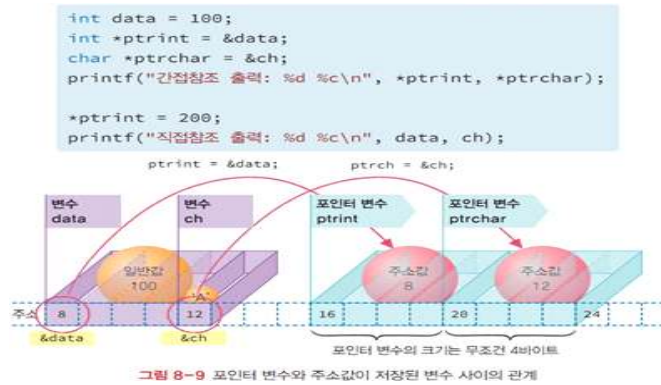
    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

변수명	주소값	저장값
data	00B5FB18	100
pprint	00B5FB0C	00B5FB18

C:\Users\김범기\source\repos\Project6\Debug\Project6.exe  
이 창을 닫으려면 아무 키나 누르세요...

## 8.2 간접 연산자 \*와 포인터 연산 .....



간접연산자(indirection operator) \*는 포인터 변수가 가리키고 있는 변수를 참조한다. \*ptntr는 포인터 ptntr가 가리키고 있는 변수 자체를 의미하며 직접참조(direct access)는 변수 data 자체를 사용해 자신을 참조하지만 간접참조(indirect access)는 \*ptntr를 이용해서 변수 data를 참조한다. 즉, int \*ptntr = &data 라고 선언했을 때 int \*ptntr; ptntr = &data; 즉, ptntr는 &data를 뜻하고, \*ptntr는 data를 뜻한다.

포인터 변수는 간단한 더하기와 뺄셈 연산에서 포인터가 가리키는 변수 크기에 비례한 포인터의 연산은 절대적인 주소의 계산이 아니며 포인터에 저장된 주소값의 연산으로 이웃한 이전 또는 이후의 다른 변수를 참조한다. int형 포인터 pi에 저장된 주소값이 100이라고 가정했을 시 (pi+1)은 101이 아니라 주소값 104이다. 즉 (pi+1)은 pi가 가리키는 다음 int형의 주소를 의미한다. int형은 4바이트의 크기를 가지고 있으므로 100다음이 104가 오는 것이다.

실습예제 8-4

```

dereference.c
포인터 변수와 간접연산자 *를 이용한 간접참조

01 // file: dereference.c
02 #include <stdio.h>
03
04 int main(void)
05 {
06     int data = 100;
07     char ch = 'A';
08     int *ptntr = &data;
09     char *ptrchar = &ch;
10     printf("간접참조 출력: %d %c\n", *ptntr, *ptrchar);
11
12     *ptntr = 200; //변수 data를 *ptntr로 간접참조하여 그 내용을 수정
13     *ptrchar = 'B'; //변수 ch를 *ptrchar로 간접참조하여 그 내용을 수정
14     printf("직접참조 출력: %d %c\n", data, ch);
15
16     return 0;
17 }
            
```

설명

06 int 형 변수 data 선언하여 100 저장

07 char 형 변수 ch 선언하여 문자 'A' 저장

08 int 형 포인터 변수 ptntr를 선언하여 변수 data의 주소값을 저장

09 char 형 포인터 변수 ptrchar를 선언하여 변수 ch의 주소값을 저장

10 int 형 포인터 변수 ptntr와 간접연산자 \*를 이용한 \*ptntr를 사용하여 data의 저장값을 출력

10 char 형 포인터 변수 ptrchar와 간접연산자 \*를 이용한 \*ptrchar를 사용하여 ch의 저장값을 출력

12 int 형 포인터 변수 ptntr와 간접연산자 \*를 이용한 \*ptntr를 사용하여 data의 저장값을 200으로 수정

13 char 형 포인터 변수 ptrchar와 간접연산자 \*를 이용한 \*ptrchar를 사용하여 ch의 저장값을 'B'로 수정

14 int 형 변수 data에 저장된 값과 char 형 변수 ch에 저장된 값을 출력

실행결과

간접참조 출력: 100 A

직접참조 출력: 200 B

다음 예제는 포인터 변수와 간접연산자 \*를 이용한 간접참조를 확인하는 프로그램이다.

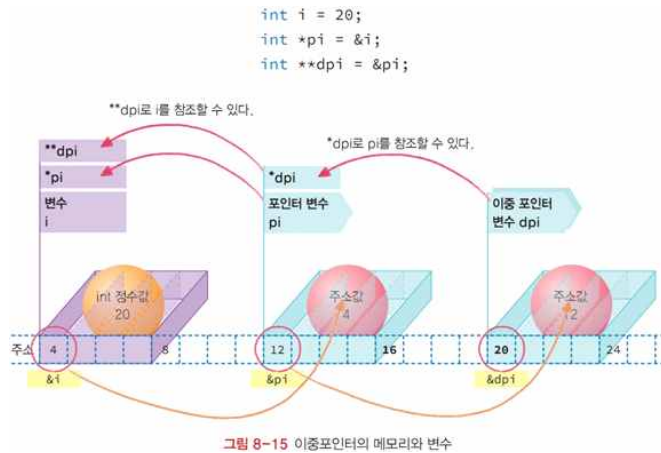
주소 연산자 &와 간접 연산자 \*, 모두 전위 연산자로 서로 반대의 역할을 한다.

주소 연산 '&변수'는 변수의 주소값이 결과값이지만 간접 연산 '\*포인터변수'는 포인터 변수가 가리키는 변수 자체가 결과값이다. '\*포인터변수'는 l-value와 r-value로 모두 사용이 가능하며 주소값인 '&변수'는 r-value로만 사용이 가능하다. '\*포인터변수'와 같이 간접연산자는 포인터 변수에만 사용이 가능하며 주소 연산자는 '&변수'와 같이 모든 변수에 사용이 가능하다.

## 8.3 포인터 형변환과 다중 포인터 .....

이중 포인터는 포인터 변수의 주소값을 갖는 변수이며 삼중 포인터는 다시 이중 포인터의 주소값을 갖는 변수이다.

다중 포인터 변수를 이용하여 일반 변수를 참조하려면 가리킨 횟수만큼 간접연산자를 이용하여야 한다. 즉 이중 포인터 변수 dpi는 \*\*dpi가 바로 변수 i를 의미하며 문장 \*pi = i + 2;는 변수 i를 2 증가시키는 문장이다. 포인터 변수 pi에서 \*pi도 변수 i이다. 문장 \*\*dpi = \*pi + 2;는 변수 i를 2 증가시키는 문장으로 해석할 수 있다.



```
*pi = i + 2;           // i = i + 2;
**dpi = *pi + 2;       // i = i + 2;
```

그림 8-16 다중포인터의 이용

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

간접 연산자 \*는 증감 연산자 ++, --와 함께 사용하는 경우에는 \*p++는 \*(p++)으로 (\*p)++와 다르다. 또한 ++\*p와 ++(\*p)는 같으며 \*\*\*p는 \*(++p)는 같다.

다음은 교과서에 나온 자료형 double로 선언된 두 x와 y에 표준입력으로 두 실수를 입력 받아 두 실수의 덧셈 결과를 출력하는 프로그램이다. 제한 사항은 두 변수 x와 y는 선언만 수행하며, 포인터 변수인 px와 py만을 사용하여 모든 과정을 코딩한다.

Lab 8-3
sumpointer.c

```

01 // file: sumpointer.c
02 #define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의
03 #include <stdio.h>
04
05 int main(void)
06 {
07     double x, y;
08     double *px = &x;
09     double *py = &y;
10
11     //포인터 변수 px와 py를 사용
12     printf("두 실수 입력: ");
13     scanf("%lf %lf", _____, _____);
14     //합 출력
15     printf("%.2f + %.2f = %.2f\n", _____, _____, _____);
16
17     return 0;
18 }

```

정답

```

13 scanf("%lf %lf", px, py);
15 printf("%.2f + %.2f = %.2f\n", *px, *py, *px + *py);

```



## 9.1 배열 선언과 초기화 .....

만약 40명 학생 성적을 변수에 저장하려면 40개의 변수를 선언해야 하는 번거로움이 이따 하지만 이때 배열(array)의 필요성이 나타나는데 여러 변수들이 같은 배열이름으로 일정한 크기의 연속된 메모리에 저장되는 구조가 있다면 편리하게 사용할 수 있으며 변수를 일일이 선언하는 번거로움을 해소할 수 있다. 배열은 동일한 크기로 지정된 배열크기만큼 확보한 연속된 저장공간이다. 배열을 구성하는 각각의 항목을 배열의 원소(elements)라고하며 배열원소는 첨자(index) 번호라는 숫자를 이용해 쉽게 접근이 가능하다.

배열선언

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

다음은 배열을 선언하는 방법이다.

배열은 대괄호 사이에 [배열크기]와 같이 기술하여 선언한다.

그림 9-2 배열선언 구문

```
int score[5];

//배열 원소에 값 저장
score[0] = 78;
score[1] = 97;
score[2] = 85;
//배열 4번째 원소에 값 저장하지 않아 쓰레기값 저장
score[4] = 91;
score[5] = 50; //문법오류는 발생하지 않으나 실행오류 발생
```

배열원소는 5개이므로 총 5 \* 4바이트 = 20바이트

4바이트

초기값이 없음

score[5]로 참조 불가능하며, 참조하면 실행 오류 발생

배열 score 전체

그림 9-3 배열선언과 원소참조

배열선언 후 배열원소에 접근하기 위해서는 첫 번째인 배열원소를 접근하는 첨자값은 0, 다음 두 번째 원소는 1 이런식으로 0부터 1씩 증가하며 배열에서 유효한 첨자의 범위는 0부터 (배열크기-1)까지이다. 배열에서 초기값은 배열을 함수내부에서 선언 후 원소에 초기값을 저장하지 않으면 가비지(garbage)가 저장된다.

//배열원소 출력

```
for (i = 0; i < SIZE; i++)
    printf("%d ", score[i]);
```

첨자가 0에서 SIZE-1까지 순회해야 하므로 이 조건을 이용함

그림 9-4 배열원소 출력을 위한 반복 구문

다음 그림은 배열 원소를 일괄 출력하는 예이다.

먼저 반복문을 통하여 배열원소에 접근한 뒤 배열원소에 접근하는 첨자값을 1씩 늘리며 배열에 담겨있는 값들을 출력 할 수 있다.

배열 선언시 초기화 방법은 배열선언을 하면서 대입연산자를 이용해야 하며 중괄호 사이에 여러 원소값을 쉼표로 구분하여 기술한다. 만약 초기값이 있으면 배열크기는 생략이 가능하지만 생략하면 자동으로 중괄호 사이에 기술된 원소 수가 배열크기가 된다. 또 배열의 일부 원소만 초기화하면 지정하지 않은 원소의 초기값은 자동으로 모두 기본값으로 저장되며 기본값이란 자료형에 맞는 0을 저장하는 것을 말한다. 즉 정수형은 0, 실수형은 0.0 그리고 문자형은 '\0'인 널 문자를 쓰레기 값으로 저장한다.

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};
```

그림 9-5 배열 초기화 구문

```
int dist[5] = {12, 23, 17};
```

자동 값 0   자동 값 0

배열크기를 지정한 후 초기값 지정 원소 수가 배열크기보다 많으면 다음의 문법오류가 발생한다.

```
int dist[5] = {12, 23, 17, 55, 57, 71};
int dist[5] = {0};
```

error C2078: 이니셜라이저가 너무 많습니다.

지정한 배열크기보다 초기값 수가 적으면 모두 0으로 채워지므로 모든 배열 원소가 0으로 채워진다.

그림 9-6 배열크기가 지정된 경우의 초기값 수



## 9.2 이차원 배열과 삼차원 배열

### 이차원 배열선언

원소자료형 배열이름 [배열행크기] [배열열크기];

배열선언 시 배열크기는 생략할 수 없으며  
배열크기는 리터럴 상수, 매크로 상수  
또는 그들의 연산식이 허용된다.

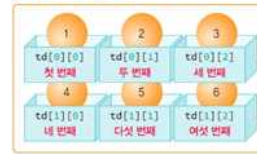
```
#define RSIZE 5
#define CSIZE 2

int score[RSIZE][CSIZE];

double point[2][3];
char ch[5][80];
float grade[7][CSIZE];
```

그림 9-11 이차원 배열선언 구문

### 행과 열 개념의 이차원 배열



개념적인 행과 열의 2차원 배열은 실제 메모리에서 한 줄만으로 연속적인 메모리에 저장공간이 확보된다.

### 실제 순차적 저장구조인 이차원 배열



그림 9-13 이차원 배열의 메모리 내부 구조

이차원 배열은 테이블 형태의 구조를 가지고 있으며 4행 2열(4 x 2)의 이차원 배열이 필요하다. 총 배열원소는 8개이며 이차원 배열 선언 시에는 이차원 배열선언은 2개의 대괄호가 필요하며 첫 번째 대괄호에는 배열의 행 크기를 두 번째는 배열의 열 크기를 지정한다. 또한 배열선언 시 초기값을 저장하지 않으면 모두 쓰레기 값이 들어가므로 반드시 행과 열의 크기 명시해야 한다.

행우선 배열은 실제로 이차원 배열이 메모리에 저장되는 모습이며 행과 열의 개념이 아니라 일차원과 같은 연속적인 메모리 공간에 저장하는 개념이다. 첫 번째 행 모든 원소가 메모리에 할당된 이후에 두 번째 행의 원소가 순차적으로 할당된다.

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

그림 9-14 이차원 배열에서 배열원소 순차적으로 출력

이차원 배열을 출력하기 위해선 외부반복을 사용하며 제어변수 i는 행을 0에서 (행의 수-1)까지 순차적으로 참조하고 내부반복 부분에서는 제어변수 j는 0에서 (열의 수-1)까지 열을 참조한다.

이차원 배열을 선언하면서 초기값을 지정하는 방법에는 2가지 방법이 있는데 첫 번째는 중괄호를 중첩되게 이용하는 방법이고 두 번째 방법은 하나의 중괄호를 사용하는 방법이다. 이차원 배열선언 초기값 지정에서도 행의 크기는 명시하지 않을 수 있다. 열 크기만 명시한다면 명시된 배열원소 수와 열 크기를 이용하여 행의 크기를 자동으로 산정되며 총 배열원소 수보다 적게 초기값이 주어지면 나머지는 모두 기본값인 0, 0.0 또는 '\0'이 저장된다.

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



그림 9-16 이차원 배열선언 초기화

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

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

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

배열원소를 순차적으로 2행 3열의 원소값으로 인지한다.

명시된 열 수인 3을 보고 3개씩 나누어보면 행이 2인 것을 알 수 있다.

그림 9-17 이차원 배열선언 초기화

```
int a[2][4] = {10, 30, 40, 50, 1, 3, 0, 0};
int a[2][4] = {10, 30, 40, 50, 1, 3};
int a[][4] = {10, 30, 40, 50, 1, 3};
int a[2][4] = { {10, 30, 40, 50}, {1, 3} };
int a[][4] = { {10, 30, 40, 50}, {1, 3} };
```

그림 9-18 이차원 배열 2행 4열에 순차적으로 10, 30, 40, 50, 1, 3, 0, 0이 저장된 초기화 문장

## 9.3 배열과 포인터의 관계

배열은 실제 포인터와 연관성이 매우 많다. 다음 배열 `score`에서 배열이름 `score` 자체가 배열 첫 원소의 주소값인 상수이다. `score`와 `&score[0]`는 같으며 배열이름 `score`를 이용하여 모든 배열원소의 주소와 저장값을 참조할 수 있다. 간접연산자를 이용한 `*score`는 변수 `score[0]`과 같고 `(score + 1)`은 `&score[1]`이다. 이것을 확장하면 `(score + i)`는 `&score[i]`이고 다시 간접연산자를 사용하면 `*(score + i)`는 `score[i]`와 같다.

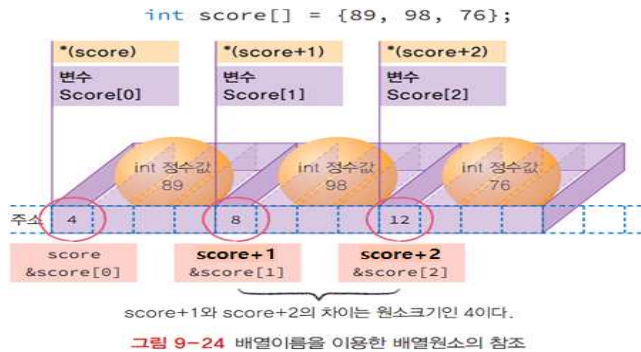


표 9-1 배열원소의 주소와 내용 값의 다양한 접근방법

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

연산식 `(score + i)`는 `score` 다음 `i`번째 원소의 주소값을 알아내는 연산식이다. 간접연산자 `*`를 사용한 연산식 `*(score + i)`에서 배열 `score`의 `(i+1)` 번째 배열원소로 `score[i]`와 같다. `(*score + 1)`은 배열 첫 번째 원소에 1을 더하는 연산식이고 연산식 `*(score+1)`과 `(*score + 1)`은 다르므로 주의해야 한다.

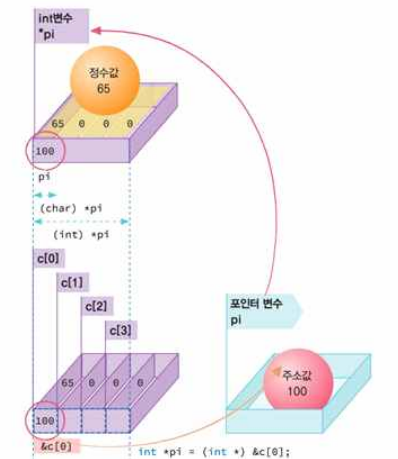
다음 그림은 포인터 변수를 이용한 배열의 원소를 참조하는 그림이다. 먼저 배열 첫 원소의 주소를 포인터에 저장한 후 주소를 1씩 증가시키면 각각의 원소를 참조할 수 있다.

포인터 `pa`에 `&a[0]`를 저장하면 연산식 `*(pa+i)`로 배열원소를 참조할 수 있고 포인터 `pa`로도 배열과 같이 첨자를 이용할 수 있다. 또한 `pa[i]`로 배열원소를 참조할 수 있다.

표 9-2 포인터와 증감연산의 다양한 연산식

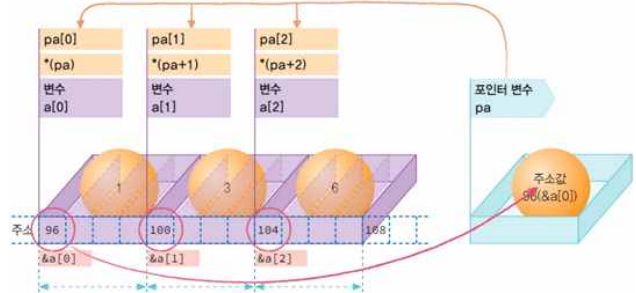
연산식	결과값	연산 후 *p의 값	연산 후 p 증가
++*p	++(*p)	*p + 1: p의 간접참조 값에 1 증가	*p가 1 증가
*p++	*(p++)	*p: p의 간접참조 값	변동 없음
--*p	--(*p)	*p - 1: p의 간접참조 값에 1 감소	*p가 1 감소
(*p)--		*p: p의 간접참조 값	*p가 1 증가

```
char c[4] = {'A', '\0', '\0', '\0'}; //문자'A' 코드값: 65
int *pi = (int *) &c[0];
printf("%d %c\n", (int) *pi, (char) *pi); //정수값 65와 문자'A'가 출력
```



```
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
```



포인터 변수는 자동으로 형변환(type cast)이 불가능하여 만약 필요하다면 명시적으로 형변환을 수행해야 한다. 자료형 `(char *)`를 `(int *)`로 변환하고 `(int *)` 형 변수 `pi`에 저장하는 것은 가능하게 하기 위해 `*pi`로 수행하는 간접참조하고 `pi`가 가리키는 주소에서부터 4바이트 크기의 `int` 형 자료를 참조하며 문자배열에 저장된 문자 4개를 그대로 4바이트인 정수 65로 참조한다. 변환된 포인터 변수는 지정된 주소값을 시작하여 그 변수 자료형의 크기만큼 저장공간을 참조한다.

## 9.4 포인터 배열과 배열 포인터 .....

일반 변수의 배열이 있듯이 포인터 배열(array of pointer)이란 주소값을 저장하는 포인터를 배열 원소로 하는 배열이다. 일반 배열 선언에서 변수 앞에 \*을 붙이면 포인터 배열 변수 선언이 된다.

```
int a = 5, b = 7, c = 9;
int *pa[3];
pa[0] = &a; pa[1] = &b; pa[2] = &c;
```

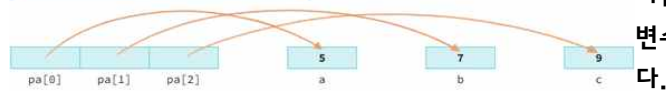


그림 9-31 포인터 배열 pa의 선언과 이해

여기서 배열 pa는 배열크기가 3인 포인터 배열이다.

다음 그림을 보면 먼저 pa[0]은 변수 a의 주소를 저장하며 pa[1]은 변수 b의 주소이고 pa[2]는 변수 c의 주소라는 것을 확인할 수 있다.

자료형 int인 일차원 배열 int a[]의 주소는 (int\*)인 포인터 변수에 저장할 수 있다. 열이 4인 이차원 배열 ary[][4]의 주소를 저장하려면 배열 포인터(pointer to array) 변수 ptr을 문장 int(\*ptr)[4];로 선언해야한다. 여기서 대괄호 사이의 4는 가리키는 이차원 배열에서의 열의 크기이다. 즉 이 차원 배열의 주소를 저장하는 포인터 변수는 열 크기에 따라 변수 선언이 달라진다.

일차원과 이차원 배열 포인터의 변수 선언

원소자료형 *변수이름;	원소자료형 (*변수이름) [ 배열크기];
변수이름 = 배열이름;	변수이름 = 배열이름;
또는	또는
원소자료형 *변수이름 = 배열이름;	원소자료형 (*변수이름) [ 배열크기] = 배열이름;

```
int a[] = {8, 2, 8, 1, 3};
int *p = a;

int ary[][4] = {5, 7, 6, 2, 7, 8, 1, 3};
int (*ptr)[4] = ary; //열이 4인 배열을 가리키는 포인터
//int *ptr[4] = ary; //포인터 배열
```

그림 9-34 배열 포인터의 변수 선언 구문

여기서 주의할 점은 선언문장 int(\*ptr)[4];에서 괄호(\*ptr)은 반드시 필요하다는 것이다. 괄호가 없는 int \*ptr[4];은 바로 전에 배운 int형 포인터 변수 4개를 선언하는 포인터 배열 선언 문장이다. int (\*ptr)[4];는 열이 4인 이차원 배열 포인터 선언 문장이다.

일차원 배열에서 직접 배열의 원소 수인 배열 크기를 계산해 볼 수 있다. 저장공간의 크기를 바이트 수로 반환하는 연산자 sizeof를 이용한 식( sizeof(배열이름) / sizeof(배열원소) )의 결과는 배열크기이다.

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

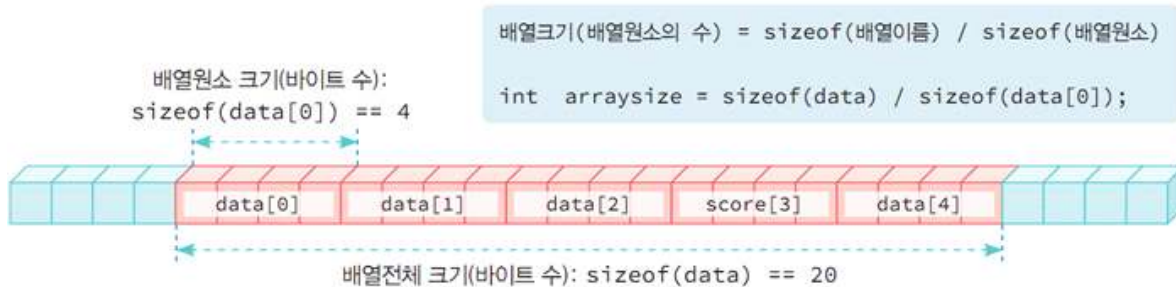


그림 9-38 배열크기인 배열원소의 수 구하기

```
//file: arraysize.c
#include <stdio.h>

int main(void) {
    int data[] = { 3,4,5,7,9 };
    printf("%d %d\n", sizeof(data), sizeof(data[0]));
    printf("일차원 배열: 배열크기 == %d\n", sizeof(data) / sizeof(data[0]));
    //4 X 3 행렬
    double x[][3] = { {1,2,3},{7,8,9},{4,5,6},{10,11,12} };
    printf("%d %d %d\n", sizeof(x), sizeof(x[0]), sizeof(x[1]), sizeof(x[0][0]));
    int rowsize = sizeof(x) / sizeof(x[0]);
    int colsize = sizeof(x[0]) / sizeof(x[0][0]);
    printf("이차원 배열: 행수 == %d 열수 == %d\n", rowsize, colsize);
    printf("이차원 배열: 전체 원소 수 == %d\n", sizeof(x) / sizeof(x[0][0]));
    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
20 4
일차원 배열: 배열크기 == 5
96 24 24
이차원 배열: 행수 == 4 열수 == 3
이차원 배열: 전체 원소 수 == 12

C:\Users\김범기\source\repos\Project6\Debug
이 창을 닫으려면 아무 키나 누르세요...
```



## 10.1 함수정의와 호출

함수(function)는 프로그램에서 특정한 작업을 처리하도록 작성한 프로그램 단위 이다. 함수는 필요한 입력을 받아 원하는 기능을 수행한 후 결과를 반환(return)하는 프로그램 단위이다.

사용자 정의 함수(user defined function)는 사용자가 필요에 의해서 개발자가 직접 개발하는 함수이다. 라이브러리 함수(library function)란 라이브러리(library) 또는 표준 함수(standard function)이다. printf()와 scanf()와 같이 이미 개발환경에 포함되어 있는 함수를 라이브러리 함수라고 하며 라이브러리 함수의 이용은 원하는 책을 도서관에서 대출 받아 이용하는 방식이라고 생각 할 수 있다. 하나의 프로그램은 라이브러리와 개발자가 만든 여러 함수로 구성되며 main() 함수의 첫 줄에서 시작하여 마지막 줄을 마지막으로 실행한 후 종료한다. 사용자가 직접 개발한 함수를 사용하기 위해서는 함수선언(function declaration), 함수호출(function call), 함수 정의(function definition)가 필요하며 필요에 따라 여러 소스 파일로 나누어 프로그래밍 할 수 있다.

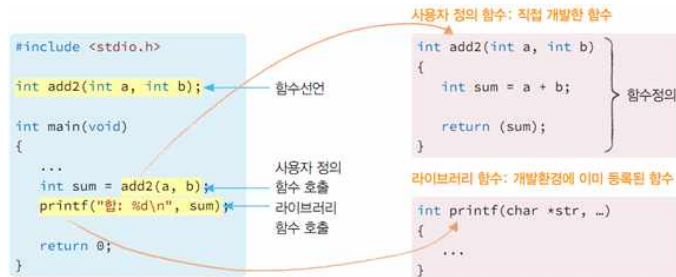


그림 10-2 함수선언, 함수호출, 함수정의

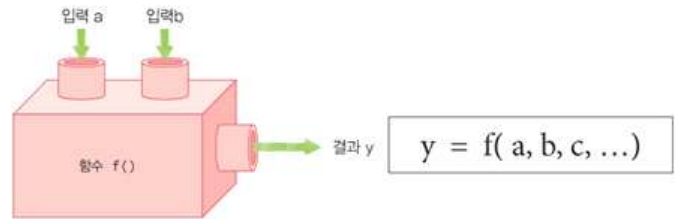


그림 10-1 함수 개념

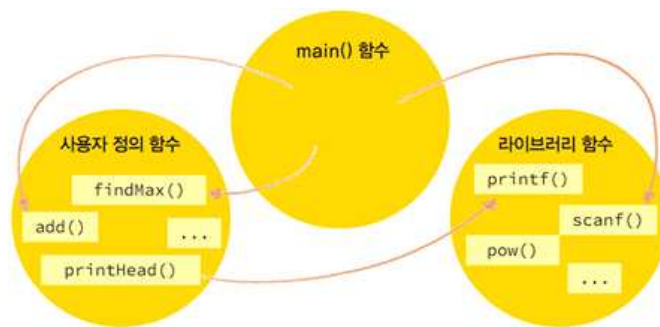


그림 10-3 절차적 프로그래밍

여러 함수로 나누어 프로그래밍하기 위해 주어진 문제를 여러 개의 작은 문제로 나누어 해결 (Devide and Conquer)한다. 또 나뉘어진 각각의 함수 단위로 구현하면 대규모 프로그램도 개발이 가능하다. 모듈화 프로그램(modular program)중 구조화된 프로그램(structured program)은 적절한 함수로 잘 구성된 프로그램이며 한번 정의된 함수는 여러 번 호출이 가능하고 소스의 중복을 최소화하여 프로그램의 양을 줄이는 효과가 있다. 절차적 프로그래밍(procedural programming)방식은 함수 중심의 프로그래밍 방식이다.

함수머리(function header)와 함수몸체(function body)로 구성되어 있으며 함수머리(function header)는 반환형이 함수 결과값의 자료형이고 함수이름은 식별자의 생성규칙을 따르며 매개변수 목록은 자료형 변수이름의 쌍으로 필요한 수만큼 콤마로 구분하여 기술한다. 또 함수몸체(function body)는 {...}와 같이 중괄호로 시작하여 중괄호로 종료하며 함수몸체에서는 함수가 수행해야 할 문장들로 구성되어있고 함수몸체의 마지막은 void 함수 제외, 결과값을 반환하는 return문장으로 종료 한다.

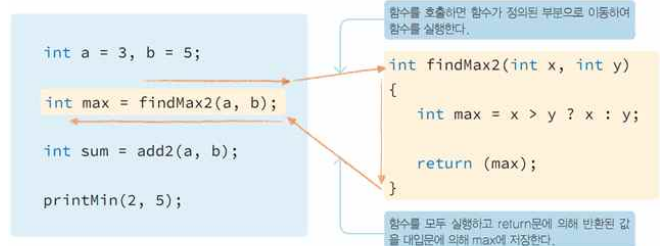


그림 10-8 함수의 호출

정의된 함수를 실행하기 위해서는 프로그램 실행 중에 함수호출

(function call)이 필요하며 함수를 호출하려면 함수이름과 함께 괄호 안에 적절한 매개변수가 필요하다. 즉 `findMax2(a, b)`, `add2(a, b)`, `printMin(3, 5)`와 같이 호출해야한다.

함수 `add2(a, b)` 또는 `findMax2(a, b)`와 같이 반환값이 있는 함수는 반환된 값을 저장하기 위해 변수(l-value)와 대입 연산자가 필요하며 문장 `sum = add2(5, 7);`은 함수 반환값 12를 변수 `sum`에 저장하는 기능을 수행한다.

함수원형(function prototype)은 함수를 선언하는 문장이다. 함수원형 구문은 함수헤더에 세미콜론을 넣은 문장이며 매개변수의 변수이름은 생략이 가능하다.

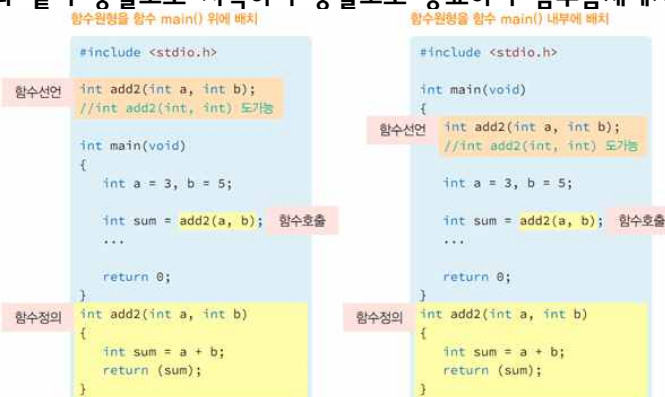


그림 10-10 함수원형 구문과 위치



## 10.2 함수의 매개변수 활용 .....

형식매개변수(formal parameters)는 함수정의에서 기술되는 매개변수 목록의 변수이다. 실매개변수(real parameters)는 함수를 호출할 때 기술되는 변수 또는 값이며 실인자(real argument) 또는 인자(argument)라고도 부른다. 실인자를 기술할 때는 함수헤더에 정의된 자료유형과 순서가 일치해야하며 실인자의 수와 자료형이 다르면 문법오류가 발생한다. 형식인자의 변수는 그 함수가 호출되는 경우에는 메모리에 할당(allocation)되고 함수가 종료, 메모리에서 자동으로 제거되며(deallocation) 형식매개변수는 함수 내부에서만 사용할 수 있는 변수이다.

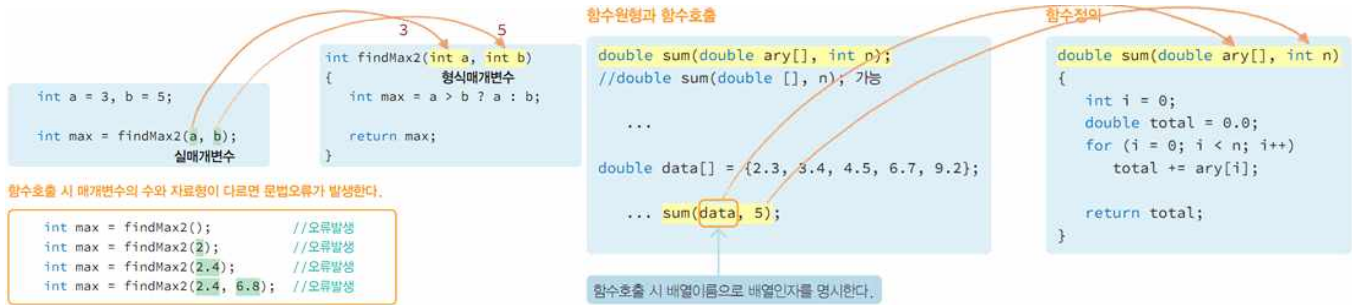


그림 10-12 형식매개변수와 실매개변수

그림 10-13 함수에서 배열 전달을 위한 함수원형과 함수호출 그리고 함수정의

실인자의 값이 형식인자의 변수에 각각 복사된 후 함수가 실행되며 매개변수의 값을 수정하더라도 함수를 호출한 곳에서의 실매개변수의 값은 변화되지 않는다. 함수의 매개변수로 배열을 전달한다면 한 번에 여러 개의 변수를 전달하는 효과가 있으며 함수 sum()은 실수형 배열의 모든 원소의 합을 구하여 반환하는 함수이다. 형식매개변수는 실수형 배열 double ary[]이며 배열크기 int n이다. 첫 번째 형식매개변수에서 배열자체에 배열크기를 기술하는 것은 무의미하며 double ary[5]보다는 double ary[]라고 기술해야한다. 실제로 함수 내부에서 실인자로 전달되는 배열크기를 알 수 없다. 그러므로 배열크기를 두 번째 인자로 사용해야하며 배열크기에 관계없이 배열 원소의 합을 구하는 함수를 만들려면 배열크기도 하나의 인자로 사용한다.

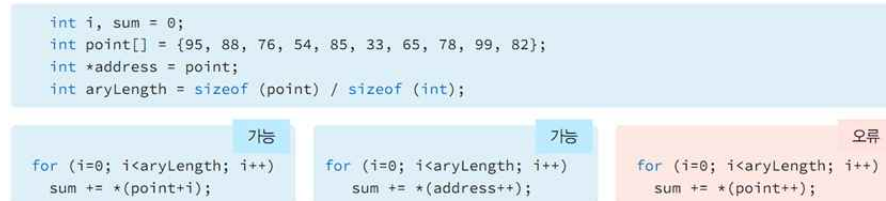


그림 10-14 간접연산자 \*를 사용한 배열원소의 참조방법

배열이름 point인 간접연산자를 사용한 배열원소의 접근 방법은 \*(point+i)이며 배열의 합을 구하려면 sum += \*(point + i); 문장을 반복해야 한다. point[10]이 배열일 때 문장은 int \*address = point; 이다. 이제 문장 sum += \*(address++)으로도 배열의 합을 구할

수 있다. 그러나 point는 주소 상수이고 sum += \*(point++)는 불가능하다. 그 이유는 증가연산식 point++의 피연산자로 상수인 point를 사용할 수 없기 때문이다.

함수에서의 배열을 전달하기 위해서는 함수헤더에 int ary[]로 기술하는 것은 int \*ary로도 대체가 가능하다. for 문의 블록은 다음 4 개 문장 모두 가능한데 먼저 변수 ary는 포인터 변수이고 주소값을 저장하는 변수이므로 증가연산자의 이용이 가능하며 그 이유는 연산식 \*ary++에서 후위 증가연산자 (ary++)의 우선순위가 가장 높기 때문이다. 또한 \*(ary++)와 같은 의미이다.



그림 10-15 함수헤더의 배열 인자와 함수정의에서 다양한 배열원소의 참조방법

## 10.3 재귀함수와 라이브러리 함수 .....

재귀 함수(recursive function)는 함수구현에서 자신 함수를 호출하는 함수이다. 또 재귀적 특성을 표현하는 알고리즘이며 재귀 함수를 이용하면 문제를 쉽게 해결할 수 있고 이해하기도 쉽다. 재귀함수에서는 수학 수식에서 재귀적 특성  $n$  계승( $n$  factorial)을 나타내는 수식  $n!$ 이다.  $1 * 2 * 3 * \dots * (n-2) * (n-1) * n!$ 을 의미하며 즉  $n!$ 의 정의에서 보듯 계승은 재귀적 특성을 갖고 있다.  $n!$ 을 구하기 위해서  $(n-1)!$ 을 먼저 구한다면 쉽게  $n!$ 을 구할 수 있으며  $(n-1)!$ 을 구하기 위해서는 다시  $(n-2)!$ 이 필요하다.  $n!$ 을 함수 `factorial(n)`로 구현한다면 함수 `factorial(n)` 구현에서 다시 `factorial(n-1)`을 호출하여 그 결과를 이용할 수 있다.

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

그림 10-18  $n!$ 의 정의와 재귀 특성

```
if (n <= 1)
    n! = 1
else
    n! = n * (n-1)!
```

```
int factorial(int num)
{
    if (num <= 1)
        return 1;
    else
        return (num * factorial(num - 1));
}
```

그림 10-19  $n!$ 을 위한 함수 `factorial()`

```
#include <stdlib.h> //rand(), srand()을 위한 헤더파일 포함
#include <time.h>    //time()을 위한 헤더파일 포함

#define MAX 100

int main(void)
{
    ...
    srand((long) time(NULL));
    ...
    number = rand() % MAX + 1;
    ...
}
```

time(NULL) 값을 인자로  
srand()를 호출한다.

1에서 MAX까지 하나의  
난수가 생성된다.

함수 `rand()`는 함수호출 순서에 따라 항상 일정한 수가 반환된다. 항상 41, 18467, 6334, 26500, 19169 값들이 출력되며 매번 난수를 다르게 생성한다. 시드(seed)값이란 난수를 다르게 만들기 위해 처음에 지정하는 수이며 시드값이 다르면 난수가 달라진다.

그림 10-22 1에서 100사이의 난수 생성 모듈

표 10-3 여러 라이브러리를 위한 헤더 파일

헤더파일	처리 작업
<code>stdio.h</code>	표준 입출력 작업
<code>math.h</code>	수학 관련 작업
<code>string.h</code>	문자열 작업
<code>time.h</code>	시간 작업
<code>ctype.h</code>	문자 관련 작업
<code>stdlib.h</code>	여러 유틸리티(텍스트를 수로 변환 등) 함수

수학 관련 함수를 사용하려면 헤더 파일 `math.h`을 삽입하여야 하며 헤더파일 `math.h`에 선언되어 있는 수학 관련 함수이다.

표 10-1 수학 관련 함수

함수	처리 작업
<code>double sin(double x)</code>	삼각함수 $\sin$
<code>double cos(double x)</code>	삼각함수 $\cos$
<code>double tan(double x)</code>	삼각함수 $\tan$
<code>double sqrt(double x)</code>	제곱근, square root( $x$ )
<code>double exp(double x)</code>	$e^x$
<code>double log(double x)</code>	$\log_e(x)$
<code>double log10(double x)</code>	$\log_{10}(x)$
<code>double pow(double x, double y)</code>	$x^y$
<code>double ceil(double x)</code>	$x$ 보다 작지 않은 가장 작은 정수
<code>double floor(double x)</code>	$x$ 보다 크지 않은 가장 큰 정수
<code>int abs(int x)</code>	정수 $x$ 의 절대 값
<code>double fabs(double x)</code>	실수 $x$ 의 절대 값

C언어에서 라이브러리 함수를 제공하며 여러 라이브러리 함수를 위한 함수원형과 상수, 그리고 매크로가 여러 헤더파일이다.

## 11.1 문자와 문자열 .....

c언어에서 문자는 영어의 알파벳 등처럼 작은 따옴표로 둘러싸서 'A'와 같이 표기한다. C 언어에서 저장공간 크기 1바이트인 자료형 char로 지원하며 작은 따옴표에 의해 표기된 문자를 문자 상수라고 한다. 문자열(string)은 문자의 모임인 일련의 문자를 뜻하며 일련의 문자 앞 뒤로 큰 따옴표로 둘러싸서 "문자열"로 표기하며 큰 따옴표에 의해 표기된 문자열을 문자열 상수라고 한다. "A"처럼 문자 하나도 큰 따옴표로 둘러싸면 문자열 상수를 뜻한다.



그림 11-1 문자 상수와 선언



그림 11-2 문자열 상수와 선언

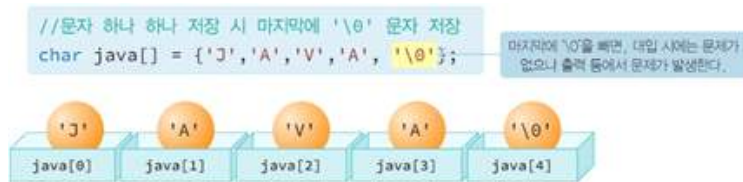


그림 11-4 문자열을 위한 문자 하나 하나의 초기화 선언

문자와 문자열 선언 시 char형 배열에 문자열을 저장하며 문자 배열을 선언하여 각각의 원소에 문자를 저장한다. 문자열의 마지막을 의미하는 NULL 문자 '\0'가 마지막에 저장되는데 이 문자는 꼭 필요하다. 문자열이 저장되는 배열크기는 반드시 저장될 문자 수보다 1이 커야 널(NULL) 문자를 문자열의 마지막으로 인식하고 문자열의 마지막에 널(NULL) 문자가 없다면 출력과 같은 문자열 처리에 문제가 발생하는 상황이 되니 주의해야 한다. 배열 선언 시 초기화 방법은 중괄호를 사용하며 문자 하나 하나를 싼표로 구분하여 입력하고 마지막 문자로 널(NULL)인 '\0'을 삽입하여 초기화 한다.

배열 선언 시 저장할 큰 따옴표를 사용해 문자열 상수를 바로 대입하며 배열 초기화 시 배열크기는 지정하지 않는 것이 더 편리하다. 지정한다면 마지막 문자인 '\0'을 고려해 실제 문자 수보다 1이 더 크게 배열크기를 지정해야 한다.

문자열 상수를 문자 포인터가 가리키는 방식은 문자 포인터 변수에 문자열 상수를 저장(사실은 문자열 상수 주소를 저장)한다. 단, 문자 포인터에 의한 선언으로는 문자 하나 하나의 수정은 불가능하니 주의해야 한다.

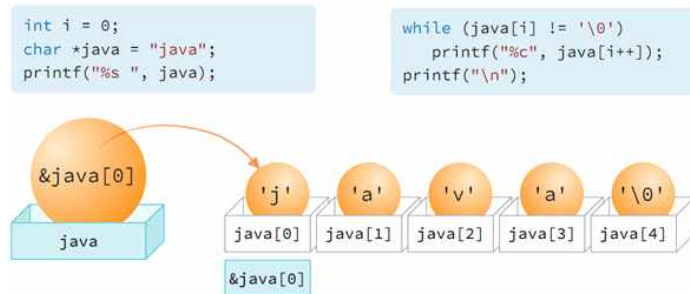


그림 11-6 문자 포인터를 사용한 문자열 처리

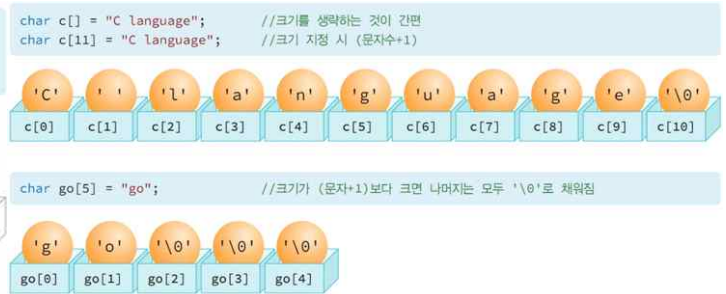


그림 11-5 문자열 상수로 문자배열 초기화

함수 gets\_s()는 한 행의 문자열 입력할 때 사용하며 사용하기 위해서는 헤더파일 stdio.h 를 삽입하여야 한다. 함수 gets\_s()는 [enter] 키를 누를 때까지 한 행을 버퍼에 저장 한 후 입력처리를 한다. 따라서 두 함수의 차이는 버퍼의 유무라고 생각할 수 있다. 함수 puts()는 한 행에 문자열을 출력할 때 사용하며 사용하기 위해서는 헤더파일 stdio.h 를 삽입하여야 한다. 또 만약 오류가 발생하면 EOF를 반환을 반환하는데 여기서 EOF란 기호 상수 EOF(End Of File)인 파일의 끝이라는 의미로 stdio.h 헤더파일에 정수 -1로 정의되어 있다.(#define EOF (-1)) 함수 gets()와 반대로 문자열의 마지막에 저장된 '\0'를 '\n'로 교체하여 버퍼에 전송하며 버퍼의 내용이 모니터에 출력되면 문자열이 한 행에 출력된다.

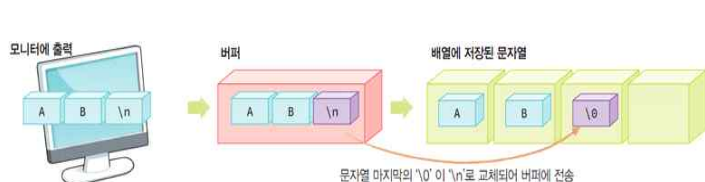


그림 11-13 함수 puts()의 처리 과정

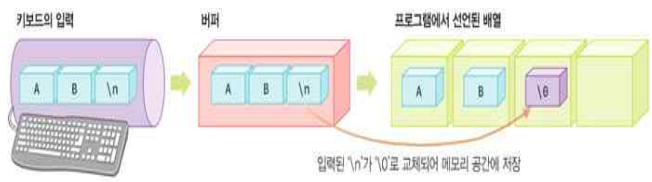


그림 11-12 함수 gets()의 처리 과정



11.2 문자열 관련 함수 .....

표 11-1 문자열 배열에 관한 다양한 함수

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

C언어에서는 헤더파일 string.h에 함수원형으로 선언된 라이브러리 함수로 제공한다. 문자열 비교와 복사, 그리고 문자열 연결 등과 같은 다양한 문자열 처리를 하기 위해 사용되며, 자료형 size\_t(비부호 정수형(unsigned int type)), 자료형 void \*(아직 정해지지 않은 다양한 포인터를 의미) 등이 있다. 다음 표는 문자열 배열에 관한 다양한 함수들이다.

함수 strlen()은 문자열의 길이를 반환하는 함수이며 함수 원형은 size\_t strlen(const char \*str)이고 size\_t는 자료형 unsigned int와 같다.하지만 사용하기 위해서는 헤더파일 string.h 를 인클루드해야 하고 문자열 관련 함수는 대부분 strxxx()로 명명한다.

문자열 복사 함수

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

- 앞 문자열 dest에 처음에 뒤 문자열 null 문자를 포함한 source를 복사하여 그 복사된 문자열을 반환한다.
- 앞 문자열은 수정되지만 뒤 문자열은 수정될 수 없다.

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

- 앞 문자열 dest에 처음에 뒤 문자열 source에서 n개 문자를 복사하여 그 복사된 문자열을 반환한다.
- 만일 지정된 maxn이 source의 길이보다 길면 나머지는 모두 널 문자가 복사된다. 앞 문자열은 수정되지만 뒤 문자열은 수정될 수 없다.

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

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

- 두 번째 인자인 sizedest는 정수형으로 dest의 크기를 입력한다.
- 반환형 errno\_t는 정수형이며 반환값은 오류번호로 성공하면 0을 반환한다.
- Visual C++에서는 앞으로 함수strcpy\_s()와 strncpy\_s()의 사용을 권장한다.

함수 strcmp()은 String Compare를 줄임말이며 문자열 비교와 복사, 그리고 문자열 연결 등과 같은 다양한 문자열 처리시 사용한다. 이 함수 역시 헤더파일 string.h에 함수원형으로 선언된 라이브러리 함수로 제공된다.  
함수 strcpy\_s()은 String Copy의 줄임말이며 함수 strcpy\_s()와 strncpy\_s()는 문자열을 복사하는 함수이다. 하지만

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

함수 strcat\_s()은 하나의 문자열 뒤에 다른 하나의 문자열을 연이어 추가해 연결하며 첫 문자열에 뒤에 두번째 문자열을 연결한다. 앞의 문자열 주소를 반환하는데 앞 인자인 dest의 저장공간이 연결된 문자열의 길이보다 부족하면 문제가 발생한다. strncat() 함수는 전달인자의 마지막에 연결되는 문자의 수를 지정한다.

문자열 연결 함수

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

- 앞 문자열 dest에 뒤 문자열 source를 연결(concatenate)해 저장하며, 이 연결된 문자열을 반환하고 뒤 문자열은 수정될 수 없다.

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

- 앞 문자열 dest에 뒤 문자열 source중에서 n개의 크기만큼을 연결(concatenate)해 저장하며, 이 연결된 문자열을 반환하고 뒤 문자열은 수정될 수 없다.
- 지정된 maxn이 문자열 길이보다 크면 null 문자까지 연결한다.

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

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

- 두 번째 인자인 sizedest는 정수형으로 dest의 크기를 입력한다.
- 반환형 errno\_t는 정수형이며 반환값은 오류번호로 성공하면 0을 반환한다.
- Visual C++에서는 앞으로 함수strcat\_s()와 strncat\_s()의 사용을 권장한다.

그림 11-18 문자열 연결 함수



## 11.3 여러 문자열 처리 .....

```
char *pa[] = {"JAVA", "C#", "C++"};
//각각의 3개 문자열 출력
printf("%s ", pa[0]); printf("%s ", pa[1]); printf("%s\n", pa[2]);
```

배열의 크기는 문자열 개수인 3을 지정하거나 빈 공백으로 한다.

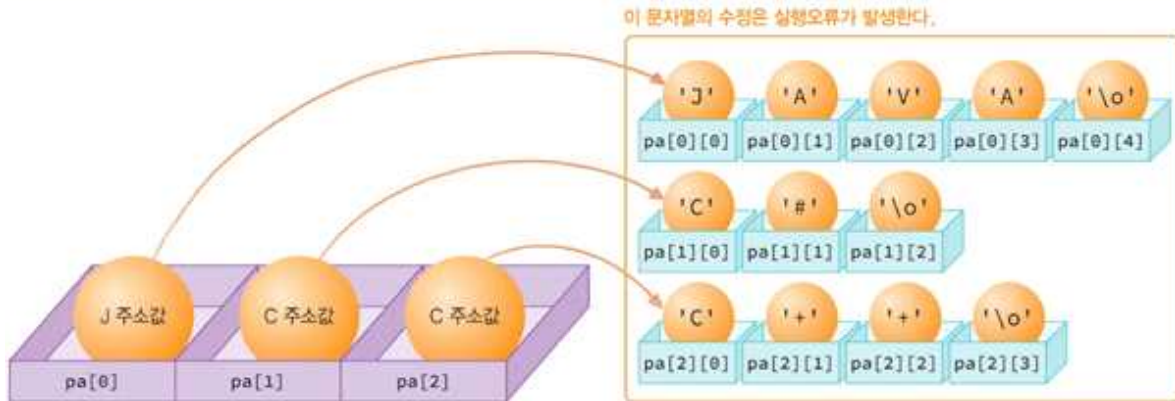


그림 11-22 문자 포인터 배열을 이용한 여러 개의 문자열 처리

문자 포인터 배열을 이용하는 방법은 여러 개의 문자열을 처리하는 하나의 방법이 있으며 문자 포인터 배열은 여러 개의 문자열을 참조가 가능하다. 각 포인터 배열 원소(문자 포인터)가 하나의 문자열을 참조할 수 있으며 문자 포인터 배열 이용 방법은 각각의 문자열 저장을 위한 최적의 공간을 사용할 수 있다는 장점이 있지만 문자 포인터를 사용해서는 문자열 상수의 수정은 불가능하다는 단점과 문장 `pa[0][2] = 'v';`와 같이 문자열의 수정은 실행오류가 발생하는 단점이 있다.

여러 개의 문자열을 처리하는 다른 방법 중에서 문자의 이차원 배열을 이용하는 방법이 있는데 이차원 배열의 열 크기는 문자열 중에서 가장 긴 문자열의 길이보다 1 크게 지정하고 가장 긴 문자열 "java"보다 1이 큰 5를 2차원 배열의 열 크기로 지정하여 사용할 수 있다. 물론 이차원 배열의 행의 크기는 문자열 수이고 3으로 지정하거나 공백으로 비워 두어야 한다. 하지만 문자의 이차원 배열에서 모든 열 수가 동일하게 메모리에 할당한다는 장점이 있지만 열의 길이가 서로 다른 경우에는 '\0' 문자가 들어가 낭비한다는 단점도 있으며 문자열을 수정할 수 있고 `ca[0][2] = 'v';`와 같이 원하는 문자 수정이 가능하다. 아래의 그림은 이차원 문자배열을 이용한 여러 문자열의 처리방법이다.

```
char ca[][5] = {"JAVA", "C#", "C++"};
//각각의 3개 문자열 출력
printf("%s ", ca[0]); printf("%s ", ca[1]); printf("%s\n", ca[2]);
```

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

이 문자열의 수정은 실행오류가 발생한다.

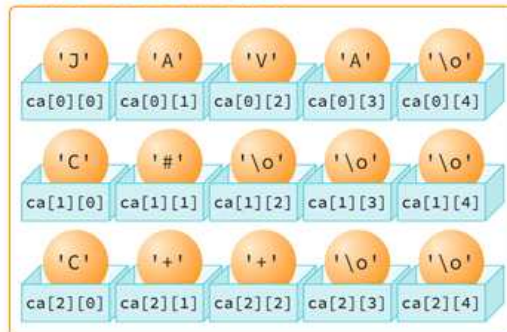


그림 11-23 이차원 문자배열을 이용한 여러 문자열 처리

## 12.1 전역변수와 지역변수

변수의 참조가 유효한 범위를 변수의 유효 범위(scope)라 한다. 변수의 유효범위는 크게 지역 유효 범위(local scope)와 전역 유효 범위(global scope)로 나눌 수 있다.

지역 유효 범위는 함수 또는 블록 내부에서 선언되어 그 지역에서 변수의 참조가 가능한 범위이며 전역 유효 범위는 다시 2가지로 나눌 수 있는데 하나의 파일에서만 변수의 참조가 가능한 범위와 프로젝트를 구성하는 모든 파일에서 변수의 참조가 가능한 범위 이렇게 2개이다.

국내전용 카드가 지역 변수라면 국내외에서 사용할 수 있는 카드는 전역변수라고 이해할 수 있다.

지역변수는 함수 또는 블록에서 선언된 변수이다. 함수나 블록에서 지역변수는 선언 문장 이후에 함수나 블록의 내부에서만 사용이 가능하다. 함수의 매개 변수도 함수 전체에서 사용 가능한 지역변수와 같으며 지역변수는 선언 후 초기화하지 않으면 쓰레기 값이 저장되므로 주의해야 한다.

```
int main(void) {
    //지역변수 선언
    int n = 10;

    //=====지역변수 n의 영역 유효 범위 =====
    printf("%d\n", n);

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

    printf("%d%d\n", n, sum); //오류발생('선언되지 않은 식별자입니다.')
    return 0;
}
```

지역변수가 할당되는 메모리 영역을 스택(stack)라고 한다.

그리고 지역변수는 선언된 부분에서 자동으로 생성되고 함수나 블록이 종료되는 순간 메모리에서 자동으로 제거된다. 이러한 이유에서 지역변수는 자동변수(automatic variable)라 한다.

전역변수(global variable)는 함수 외부에서 선언되는 변수이다. 전역변수는 외부 변수라고도 부르며 일반적으로 프로젝트의 모든 함수나 블록에서 참조할 수 있다. 다음은 전역변수 PI의 선언과 사용에 대한 예제이다.

```
// file: globalvar.c
#include <stdio.h>

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

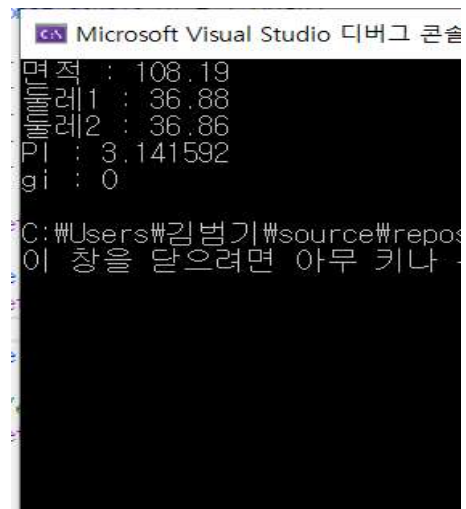
//전역변수 선언
double PI = 3.14;
int gi;

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 기본값

    return 0;
}

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



동일한 파일에서도 extern을 사용해야 하는 경우가 발생할 수 있다. 전역변수의 선언 위치가 변수를 참조하려는 위치보다 뒤에 있는 경우 전역변수를 사용하기 위해서는 extern을 사용한 참조 선언이 필요하다.

전역변수는 어디에서든지 수정할 수 있으므로 사용이 편한 장점이 있다. 그러나 전역변수에 예상하지 못한 값이 저장된다면 프로그램 어느 부분에서 수정이 되었는지 알기 어려운 단점이 있다.

이러한 문제로 전역변수는 가능한 제한적으로 사용하는 것이 바람직하다.

## 12.2 정적 변수와 레지스터 변수 .....

변수는 4가지의 기억 부류(storage class)인 auto, register, static, extern에 따라 할당되는 메모리 영역이 결정되고 메모리의 할당과 제거 시기가 결정된다. 기억부류 auto와 register는 지역변수에만 이용이 가능하고 static은 지역과 전역 모든 변수에서 이용이 가능하다. 그리고 마지막으로 extern은 전역변수에만 사용이 가능하다.

레지스터 변수는 변수의 저장공간이 일반 메모리가 아니라 CPU내부의 레지스터(register)에 할당되는 변수이다. 키워드 register를 자료형 앞에 넣어 선언하며 지역변수에만 이용이 가능하다. 또한 CPU내부에 있는 기억장소이므로 일반 메모리보다 빠르게 참조 가능하다는 장점이 있고 일반 메모리에 할당되는 변수가 아니므로 주소연산자 &는 사용이 불가능하다.

```
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.

```
#include <stdio.h>

int a = 1;
static s = a; //오류

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

    return 0;
}
```

```
#include <stdio.h>

int a = 1;
static s = 1;

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

    return 0;
}
```

static은 변수 선언에서 자료형 앞에 키워드 static 기술하여 사용하며 정적 지역변수(static global variable)와 정적 전역변수(static local variable)로 구분한다. 정적(지역) 변수의 특성으로는 초기 생성된 이후 메모리에서 제거되지 않으므로 지속적으로 저장값을 유지하며 수정 가능한 특성이 있고 프로그램이 시작되면 메모리에 할당되고, 프로그램이 종료되면 메모리에서 제거한다.

초기 값을 지정하지 않으면 자동으로 자료형에 따라 0이나 '\0' 또는 NULL 값이 저장되며 빌드 과정에서 초기화되고 다음과 같은 변수 선언 시 초기화는 수행되지 않는다. 정적 지역변수의 초기화는 반드시 상수만 가능하며 소스에서 정적 변수의 초기값에 변수를 대입하는 방식이다.

```
// file : staticlocal.c
#include <stdio.h>

void increment(void); //함수원형
```

```
int main(void) {
    //자동 지역변수
    for (int count = 0; count < 3; count++)
        increment(); //함수 3번 호출
}
```

```
void increment(void) {
    static int sindex = 1; //정적 지역변수
    auto int aindex = 1; //자동 지역변수

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

정적 지역변수는 함수나 블록에서 정적으로 선언되는 변수이며 지역변수 특성인 유효 범위는 선언된 블록 내부에서만 참조가 가능하고 전역변수 특성인 함수나 블록을 종료 해도 메모리에서 제거되지 않고 계속 메모리에 유지 관리되는 특성이 있다.

또한 변수 유효 범위(scope)는 지역변수와 같으나 할당된 저장 공간은 프로그램이 종료 되어야 메모리에서 제거되는 전역변수의 특징을 갖는다.

```
Microsoft Visual Studio 디버그 콘솔

정적 지역변수 sindex: 1,   자동 지역변수 aindex: 1
정적 지역변수 sindex: 2,   자동 지역변수 aindex: 1
정적 지역변수 sindex: 3,   자동 지역변수 aindex: 1

C:\Users\김범기\source\repos\lab12\Debug\Project12-5.exe(프로세스
이 창을 닫으려면 아무 키나 누르세요...
```

정적 전역변수는 함수 외부에서 정적으로 선언되는 변수이다. 정적 전역변수는 선언된 파일 내부에서만 참조가 가능한 변수이다. 즉 정적 전역변수는 extern에 의해 다른 파일에서 참조가 불가능하다.

```
// file : staticvar.c
#include <stdio.h>

//정적 전역변수 선언
static int svar;
//전역변수 선언
int svar;

//함수원형
void increment();
void testglobal();
// void teststatic();
```

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

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

```
//함수 구현
void increment() {
    svar++;
}
```

```
Microsoft Visual Studio 디버그 콘솔

함수 increment()가 총 5번 호출 되었습니다.
전역 변수 : 10

C:\Users\김범기\source\repos\lab12\Debug\Project12-5.exe(프로세스
이 창을 닫으려면 아무 키나 누르세요...
```



## 12.3 메모리 영역과 변수 이용 .....

메인 메모리의 영역은 프로그램 실행 과정에서 데이터(data)영역, 힙(heap) 영역, 스택(stack) 영역 세 부분으로 나뉜다. 이러한 메모리 영역은 변수의 유효범위(scope)와 생존기간(life time)에 결정적 역할을 하며, 변수는 기억부류(storage class)에 따라 할당되는 메모리 공간이 달라진다.

데이터 영역은 전역변수와 정적변수가 할당되는 저장공간이며 메모리 주소가 낮은 값에서 높은 값으로 저장 장소가 할당되고 프로그램이 시작되는 시점에 정해진 크기대로 고정된 메모리 영역이 확보된다. 힙 영역은 동적 할당(dynamic allocation) 되는 변수가 할당되는 저장공간이며 데이터 영역과 스택 영역 사이에 위치한다. 스택 영역은 함수 호출에 의한 형식 매개변수 그리고 함수 내부의 지역변수가 할당되는 저장공간이며 힙 영역과 스택영역은 프로그램이 실행되면서 영역 크기가 계속적으로 변하고 메모리 주소가 높은 값에서 낮은 값으로 저장 장소가 할당 된다. 마지막으로 코드 영역은 프로그램 코드가 저장된 공간이다.

표 12-2 변수의 종류

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

일반적으로 전역변수의 사용을 자제하고 지역변수를 주로 이용한다.

그러나 다음의 경우에는 그 특성에 맞는 변수를 이용하는데 실행 속도를 개선하고자 하는 경우에는 레지스터 변수를 사용, 함수나 블록 내부에서 함수나 블록이

종료되더라도 계속적으로 값을 저장하기 위해서는 정적 지역변수를 사용, 해당 파일 내부에서만 변수를 공유하고자 하는 경우에는 정적 전역변수, 프로그램의 모든 영역에서 값을 공유하고자 하는 경우에는 전역변수를 사용한다. 하지만 가능하면 전역 변수의 사용을 줄이는 것이 프로그램의 이해를 높일 수 있으며 발생할 수 있는 프로그램 문제를 줄일 수 있다.

다음 예제는 storage class.c와 out.c로 구성되는 프로그램이다. 이 예제는 전역변수 global, sglobal 그리고 지역변수 fa, fs의 선언과 사용을 알아보는 프로그램이다. 특히 sglobal은 정적 전역변수이므로 외부에서 참조할 수 없으며, 정적 지역변수인 fs는 함수가 종료되더라도 그전에 저장된 값이 계속 유지된다는 것을 알 수 있다.

### storageclass.c

```
// file : storageclass.c
#include <stdio.h>

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

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

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

### out.c

```
// file : out.c
#include <stdio.h>

void outfunction() {
    extern int global, sglobal;

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

    //외부 파일에 선언된 정적 전역변수이므로 실행 시 오류
    //printf("%d\n", ++sglobal);
}
```

### 실행 결과:

Microsoft Visual Studio 디버그 콘솔

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

C:\Users\김범기\source\repos\lab1\
디버깅이 중지될 때 콘솔을 자동으로
```



## 13.1 구조체와 공용체

정수나 문자, 실수나 포인터 그리고 이들의 배열등을 묶어 하나의 자료형으로 이용하는 것이 구조체이다.

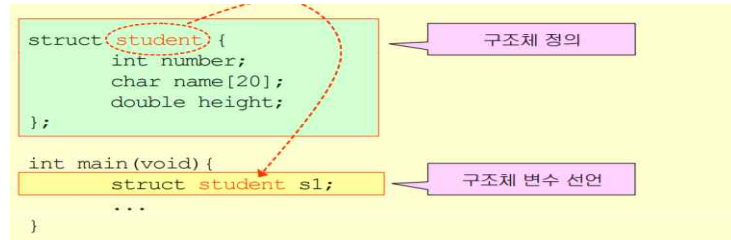
연관성이 있는 서로 다른 개별적인 자료형의 변수들을 하나의 단위로 묶은 새로운 자료형을 구조체(structure)라고 한다.

구조체는 연관된 멤버로 구성되는 통합 자료형으로 대표적인 유도 자료형이다. 즉 기존 자료형으로 새로이 만들어진 자료형을 유도 자료형(derived data types)이라 한다.

### 구조체 선언 방식

```
struct 구조체_태그_이름 {  
    자료형 멤버_이름;  
    자료형 멤버_이름;  
    ...  
};
```

### 구조체 변수 선언 방식



구조체 변수의 초기화를 위해서는 초기화 값은 다음과 같이 중괄호 내부에서 구조체의 각 멤버 정의 순서대로 초기값은 쉼표로 구분하여 기술하고 배열과 같이 초기값에 기술되지 않은 멤버값은 자료형에 따라 기본값인 0, 0, 0, '\0' 등으로 저장한다.

**struct 구조체 태그이름 변수명 = {초기값1, 초기값2, 초기값3, ...};** 과 같은 방식으로 구조체 변수를 초기화 할 수 있다.

또한 선언된 구조체형 변수는 접근 연산자 .을 사용하여 멤버를 참조할 수 있다.**ex)구조체변수이름.멤버**

### 예제코드

```
1 // file : structstudent.c  
2 #define _CRT_SECURE_NO_WARNINGS  
3 #include <stdio.h>  
4 #include <string.h>  
5  
6 int main(void) {  
7     //학생을 위한 구조체  
8     struct student {  
9         int snum; //학번  
10        int* dept; //학과 이름  
11        char name[12]; //학생 이름  
12    };  
13    struct student hong = { 201800001, "컴퓨터정보공학과", "홍길동" };  
14    struct student na = { 201800002 };  
15    struct student bae = { 201800003 };  
16  
17    //학생이름 입력  
18    scanf("%s", na.name);  
19    //na.name = "나한국"; //오류  
20    //scanf("%s", na.dept); //오류  
21  
22    na.dept = "컴퓨터정보공학과";  
23    bae.dept = "기계공학과";  
24    memcpy(bae.name, "배상문", 7);  
25    strcpy(bae.name, "배상문");  
26    strcpy_s(bae.name, 7, "배상문");  
27  
28    printf("[%d %s %s]\n", hong.snum, hong.dept, hong.name);  
29    printf("[%d %s %s]\n", na.snum, na.dept, na.name);  
30    printf("[%d %s %s]\n", bae.snum, bae.dept, bae.name);  
31  
32    struct student one;  
33    one = bae;  
34    if (one.snum == bae.snum)  
35        printf("학번이 %d로 동일 합니다.\n", one.snum);  
36    if (one.snum == bae.snum && strcmp(one.name, bae.name) && strcmp(one.dept, bae.dept))  
37        printf("내용이 같은 구조체입니다.\n");  
38  
39    return 0;  
40 }  
41
```

### 실행결과

공용체(union)는 서로 다른 자료형의 값을 동일한 저장공간에 저장하는 자료형이다.

만약 주차장이 하나있고 SUV와 세단이 각각 한 대씩 있을 때 동시에 주차는 불가능하지만 한 대의 주차는 가능하다.

이러한 겸용주차장과 비슷한 개념이 공용체이다. 공용체는 동일한 저장 장소에 여러 자료형을 저장하는 방법으로, 공용체를 구성하는 멤버에 한 번에 한 종류만 저장하고 참조할 수 있다.

```
#include <stdio.h>

union example {
    int i;
    char c;
};

int main(void)
{
    union example v;
    v.c = 'A';
    printf("v.c:%c v.i:%i\n", v.c, v.i);

    v.i = 10000;
    printf("v.c:%c v.i:%i\n", v.c, v.i);
}
```

공용체 선언

공용체 변수 선언.

char 형으로 참조.

int 형으로 참조.

```
v.c:A v.i:65
v.c: v.i:10000
```

공용체 변수의 크기는 멤버 중 가장 큰 자료형의 크기로 정해진다. 또한 공용체의 멤버는 모든 멤버가 동일한 저장공간을 사용하므로 동시에 여러 멤버의 값을 동시에 저장하여 이용할 수 없으며, 마지막에 저장된 단 하나의 멤버 자료값만을 저장한다. 공용체도 구조체와 같이 typedef를 이용하여 새로운 자료형으로 정의할 수 있다. 공용체의 초기화 값은 공용체 정의 시 처음 선언한 멤버의 초기 값으로만 저장이 가능하다.

## 예제코드

```
1 // file : union.c
2 #include <stdio.h>
3
4 union data
5 {
6     char ch; //문자형
7     int cnt; //정수형
8     double real; //실수형
9 }data1; //data1은 전역변수
10
11 int main(void) {
12     union data data2 = { 'A' };
13     // union data data2 = { 10,3 }; //컴파일 시 경고 발생
14     union data data3 = data2;
15
16     printf("%d %d\n", sizeof(union data), sizeof(data3));
17
18     //멤버 ch에 저장
19     data1.ch = 'a';
20     printf("%c %d %f\n", data1.ch, data1.cnt, data1.real);
21     //멤버 cnt에 저장
22     data1.cnt = 100;
23     printf("%c %d %f\n", data1.ch, data1.cnt, data1.real);
24     //멤버 real에 저장
25     data1.real = 3.156759;
26     printf("%c %d %f\n", data1.ch, data1.cnt, data1.real);
27
28     return 0;
29 }
```

## 실행결과

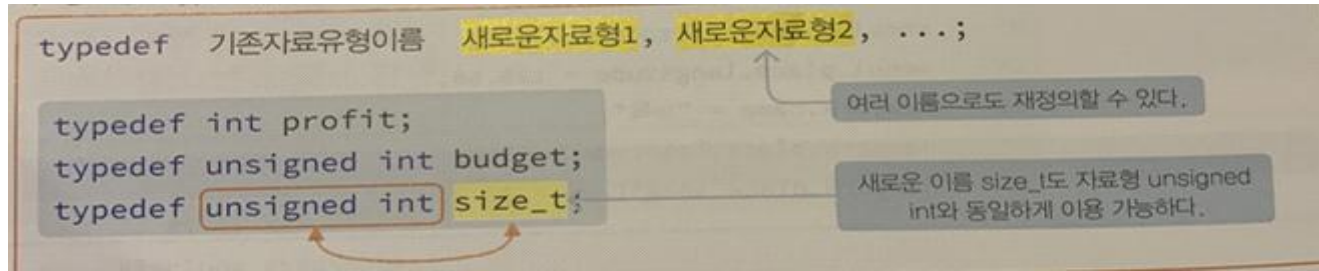
```
Microsoft Visual Studio 디버그 콘솔
8 8
a 97 0.000000
d 100 0.000000
N -590162866 3.156759

C:\Users\김범기\source\repos\Project13\
이 창을 닫으려면 아무 키나 누르세요...
```

## 13.2 자료형 재정의

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

문장 typedef int profit;은 profit을 int와 같은 자료형으로 새롭게 정의하는 문장이다.



일반적으로 자료형을 재정의하는 이유는 프로그램의 시스템 간 호환성과 편의성을 위해 필요하다.

**Visual C++:** 4byte(int salary=2000000;) → **TurboC++:** 2byte(int salary = 2000000;) 오버플로 발생

만일 이 소스를 터보 C++에서 컴파일 한다면 typedef 문장에서 int를 long형으로 수정한다면 아무 문제 없이 다른 소스는 수정 없이 그대로 이용이 가능하다.(Visual: typedef int myint; → Turbo: typedef long myint;)

```
typedef int integer, word;
```

```
integer myAge; //int myAge와 동일
```

```
word yourAge; //int yourAge와 동일
```

문장 typedef도 일반 변수와 같이 그 사용 범위를 제한한다. 그러므로 함수 내부에서 재정의되면 선언된 이후의 그 함수에서만 사용이 가능하다.

구조체 자료형은 struct date처럼 항상 키워드 struct를 사용하지 않아도 된다. typedef를 사용하여 구조체를 한 단어의 새로운 자료형으로 정의하면 이러한 불편을 덜 수 있다. 구조체 struct date가 정의된 상태에서 typedef를 사용하여 구조체 struct date를 date로 재정의 할 수 있다.

새로운 자료유형 date의 재정의와 이용

```
struct date
{
    int year;    //년
    int month;   //월
    int day;     //일
};
typedef struct date date;
```

구조체 정의와 typedef를 함께 이용한 자료형의 정의

```
typedef struct
{
    char title[30];    //제목
    char company[30];  //제작회사
    char kinds[30];    //종류
    date release;      //출시일
} software; //software는 변수가 아니라 새로운 자료형이다.
```

문장 typedef를 이용하여 구조체의 자료유형을 다른 이름으로 재정의 하여 이용하는 예제코드

```
1 // file : typedefstruct.c
2 #include <stdio.h>
3
4 typedef struct _date
5 {
6     int year;    //년
7     int month;   //월
8     int day;     //일
9 } date;
10
11 //struct date 유형을 간단히 date 형으로 사용하기 위한 구문
12 //typedef struct date date;
13
14 int main(void) {
15     //구조체를 정의하면서 바로 자료형 software로 정의 하기 위한 구문
16     typedef struct
17     {
18         char title[30];    //제목
19         char company[30];  //제작회사
20         char kinds[30];    //종류
21         date release;      //출시일
22     } software;
23
24     software vs = { "비주얼스튜디오 커뮤니티", "MS", "통합개발환경", { 2018, 8, 29 } };
25
26     printf("제품명: %s\n", vs.title);
27     printf("회사: %s\n", vs.company);
28     printf("종류: %s\n", vs.kinds);
29     printf("출시일: %d, %d, %d\n", vs.release.year, vs.release.month, vs.release.day);
30
31     return 0;
32 }
```

Microsoft Visual Studio 디버그 콘솔

```
제품명: 비주얼스튜디오 커뮤니티
회사: MS
종류: 통합개발환경
출시일: 2018, 8, 29
```

```
C:\Users\김범기\source\repos\Pro
이 창을 닫으려면 아무 키나 누르세
```

### 13.3 구조체와 공용체의 포인터 배열 .....

포인터는 각각의 자료형 저장공간의 주소를 저장하듯이 구조체 포인터는 구조체의 주소값을 저장할 수 있는 변수이다.

```

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

typedef struct lecture lecture;
lecture* p;
    
```

구조체 포인터의 변수의 선언은 일반 포인터 변수 선언과 동일하다.  
 다음은 대학 강좌를 처리하는 구조체 자료형 lecture를 선언한 구문이다.  
 구조체 포인터 변수 p는 lecture \* p로 선언된다.

구조체 포인터 멤버의 접근연산자 ->는 p->name과 같이 사용한다. 연산자 p->name은 포인터 p가 가리키는 구조체 변수의 멤버 name을 접근하는 연산식이다. 연산식 \*p.name은 접근연산자(.)가 간접연산자(\*)보다 우선순위가 높으므로 \*(p.name)과 같은 연산식이다.

접근 연산식	구조체 변수 os와 구조체 포인터 변수 p인 경우의 의미
p->name	포인터 p가 가리키는 구조체의 멤버 name
(*p).name	포인터 p가 가리키는 구조체의 멤버 name
*p.name	*(p.name)이고 p가 포인터 이므로 p.name는 문법오류가 발생
*os.name	*(os.name)를 의미하며, 구조체 변수 os의 멤버 포인터 name이 가리키는 변수로, 이 경우에는 구조체 변수 os 멤버 강좌명의 첫 문자. 다만 한글인 경우에는 오류가 발생
*p->name	*(p->name)을 의미하며, 포인터 p가 가리키는 구조체의 멤버 name이 가리키는 변수로 이 경우는 구조체 포인터 p가 가리키는 구조체의 멤버 강좌명의 첫 문자, 마찬가지로 한글인 경우 오류

```

union data
{
    char ch;
    int ont;
    double real;
}value,*p; //변수 value는 union data형이며 p는 union data 포인터 형으로 선언
p = &value; //포인터 p에 value의 주소값을 저장
p->ch = 'a'; //value.ch='a'; 와 동일한 문장
    
```

공용체 변수도 포인터 변수 사용이 가능하며, 공용체 포인터 변수로 멤버를 접근하려면 접근연산자->를 이용한다.  
 다음은 공용체 변수 value를 가리키는 포인터 p를 선언하여 p가 가리키는 공용체 멤버 ch에 'a'를 저장하는 소스이다.

구조체 배열 변수 선언은 다른 배열과 같이 동일한 구조체 변수가 여러 개 필요하면 구조체 배열을 선언하여 이용할 수 있다. 문장 lecture \*p = c;와 같이 구조체 배열이름은 구조체 포인터 변수에 대입이 가능하다. 이제 구조체 포인터 변수 p를 이용한 p[i]로 배열원소 접근이 가능하다. 다음은 구조체 배열을 선언 후 출력 처리하는 예제이다.

```

1 // file : structarray.c
2 #include <stdio.h>
3
4 typedef struct _lecture
5 {
6     char name[20]; //강좌명
7     int type; //강좌구분
8     int credit; //학점
9     int hours; //시수
10 } lecture;
11
12 char* lectype[] = { "교양", "일반선택", "전공필수", "전공선택" };
13 char* head[] = { "강좌명", "강좌구분", "학점", "시수" };
14
15 int main(void) {
16     lecture course[] = { {"인간과 사회",0,2,2},
17                          {"경제학개론",1,3,3},
18                          {"자료구조",2,3,3},
19                          {"모바일프로그래밍",2,3,4},
20                          {"고급 C프로그래밍",3,3,4} };
21
22     int arysize = sizeof(course) / sizeof(course[0]);
23
24     printf("배열크기: %d\n", arysize);
25     printf("%12s %12s %6s %6s\n", head[0], head[1], head[2], head[3]);
26     printf("=====");
27     for (int i = 0; i < arysize; i++)
28         printf("%16s %10s %5d %5d\n", course[i].name, lectype[course[i].type], course[i].credit, course[i].hours);
29
30     return 0;
31 }
    
```

배열크기: 5

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

C:\Users\김범기\source\repos\Project13\De  
 이 창을 닫으려면 아무 키나 누르세요...