C 프로그래밍

포트폴리오

컴퓨터정보공학과

20164108 권경은

CHAPTER 01 프로그래밍 언어 개요

1.1 '프로그램'이 무엇일까?

1) 스마트폰과 컴퓨터에서의 프로그램

스마트폰에서 이용하는 각종 앱이나 컴퓨터에서 작업하는 아래한글 등이 바로 프로그램(program)이다. 즉 프로그램은 스마트폰과 노트북, 혹은 데스크탑 컴퓨터에서 특정 작업을 위해 컴퓨터를 작동시키는 것이다.

스마트폰에서 사용되는 프로그램은 특정 목적의 작업을 수행하기 위한 관련 파일의 모임이며 간단히 앱 또는 어플이라 불린다.

프로그램이란 용어의 공통적 의미는 특정한 목적을 수행하기 위한 이미 정해놓은 순서적인 계획이나 절차를 의미한다. 정보기술 분야에서 프로그램은 특정 작업을 수행하기 위하여 그 처리 방법과 순서를 기술한 명령어와 자료로 구성되어 있다. 즉 프로그램은 컴퓨터에게 지시할 일련의 처리 작업 내용을 담고 있고, 사용자의 프로그램 조작에 따라 컴퓨터에게 적절한 명령을 지시하여 프로그램이 실행된다.

2) 프로그래머와 프로그래밍 언어

프로그래머는 컴퓨터와 스마트폰 등의 정보기기에서 사용되는 프로그램을 만드는 사람이다.

프로그래머는 특정 문제를 해결하기 위해 컴퓨터에게 지시할 일련의 명령을 적절히 기술하여 프로그램을 제작한다. 즉 프로그래머가 프로그램을 개발하기 위해 사용하는 언어가 프로그래밍 언어이다.

프로그래밍 언어는 사람과 컴퓨터가 서로 의사 교환을 하기 위한 언어이며, 사람이 컴퓨터에게 지시할 명령어를 기술하기 위해 만들어진 언어이다. 프로그래밍 언어로는 FORTRAN, BASIC, C, C++, JAVA, Python 등 매우 다양하다.

1.2 언어의 계층과 번역

1) 하드웨어와 소프트웨어

컴퓨터는 고철 덩어리인 하드웨어(hardware)와 이 하드웨어를 작동하게 하는 소프트웨어(software)로 구성된다.

하드웨어의 중요한 구성 요소로는 중앙처리장치(CPU), 주기억장치,(main memory), 보조기억장치(second-ary memory), 입력장치, 출력장치가 있다. 중앙처리장치는 연산을 수행하는 연산장치와 연산을 제어하는 제어장치로 구성되며 이 중앙처리장치의 칩을 프로세서라 한다.

소프트웨어는 컴퓨터가 수행할 작업을 지시하는 전자적 명령어들의 집합으로 구성된 프로그램을 말한다. 소프트웨어는 크게 응용 소프트웨어와 시스템 소프트웨어로 나뉘는데, 시스템 소프트웨어는 하드웨어 작동에 필수적이며 컴퓨터가 잘 작동하도록 도와주는 기본 소프트웨어를 말하며, 응용 소프트웨어는 문서 작성이나 인터넷 검색, 동영상 보기 등과 같은 특정한 업무에 활용되는 소프트웨어를 말한다. 시스템 소프트웨어는 운영체제(OS)와 유틸리티 프로그램으로 구분할 수 있다.

운영체제는 컴퓨터 하드웨어 장치의 전반적인 작동을 제어하고 조정하며, 사용자가 최대한 컴퓨터를 효율적으로 사용할 수 있도록 돕는 시스템 프로그램이다. 컴퓨터의 운영체제로는 리눅스, 윈도우, 맥 OS X 등이 있다. 유틸리티 프로그램은 운영체제를 돕고 컴퓨터 시스템이 원활하게 작동하도록 돕는다.

2) 기계어와 컴파일러

컴퓨터는 1 과 0 으로 표현되는 기계어 만을 인식할 수 있다. 그러므로 사람이 프로그래밍 언어로 컴퓨터에게 명령을 내리기 위해서는 프로그래밍 언어를 기계어로 변환하는 통역사인 컴파일러(compiler)가 필요하다.

기계어는 컴퓨터가 직접 이해할 수 있는 유일한 언어로, 컴파일러라는 변환기에 의해 프로그램이 기계어로 구성된 기계 코드로 변환되어 특정한 플랫폼에서 실행된다.

3) 어셈블리어

어셈블리어(assembly language)는 기계어를 프로그래머인 사람이 좀 더 이해하기 쉬운 기호 형태로 일대일 대응시킨 프로그래밍 언어이다. 어셈블리 명령어의 예로는 LDA(LoaD Address), ADD(ADD), STA(Store Address) 등이 있다.

4) 저급언어와 고급언어

컴퓨터 중앙처리장치(CPU)에 적합하게 만든 기계어와 어셈블리 언어를 모두 저급언어(Low level language)라 한다. 이와 반대로 컴퓨터의 CPU 에 의존하지 않고 우리 사람이 보다 쉽게 이해할 수 있도록 만들어진 언어를 고급언어(High level language)라 한다.

고급언어는 일상 생활에서 사용하는 수식과 간단한 영어, 그리고 다양한 기호 문자를 사용하여 만든 프로그래밍 언어를 말한다. 컴퓨터의 CPU 의 종류와 관계없이 프로그래밍 언어를 이용할 수 있는 장점이 있다.

1.3 왜 C 언어를 배워야 할까?

1) C 언어의 역사

c 언어는 1972 년 데니스 리치가 개발한 프로그래밍 언어이다. 당시 벨 연구소에서는 운영체제인 유닉스를 개발하기 위해 C 언어를 개발하였다. 어셈블리 언어 정도의 속도를 내며, 좀 더 쉽고, 서로 다른 cpu 에서도 작동되는 프로그래밍 언어가 필요했으며, 이를 위해 만든게 C 언어이다.

2) C 언어의 특징

- 절차지향 언어

  - C 언어는 함수 중심으로 구현되는 절차지향 언어이다. 절차지향 언어란 시간의 흐름에 따라 정해진 절차를 실행한다는 의미로 C 언어는 문제의 해결 순서와 절차의 표현과 해결이 쉽도록 설계된 프로그램 언어이다.

- 간결하고 효율적인 언어

  - 함수의 재귀호출이 가능하고 세세한 부분까지 제어할 수 있다. 포인터와 메모리 관리 기능을 갖고 있으며, 실행속도가 빠르다

- 이식성이 좋은 프로그래밍 언어

  - C 언어는 다양한 CPU 의 플랫폼의 컴파일러를 지원하기 때문에 이식성이 좋다.

- 다소 어렵다는 단점

  - C 언어의 단점은 다소 배우기 어렵다. 그러나 한번 익히면 다른 프로그래밍 언어 습득에 많은 도움을 준다

1.4 프로그래밍의 자료 표현

1) 프로그래밍의 내부 표현, 0 과 1

수에서 하나의 자릿수에 사용하는 숫자가 0~9 까지 열 개 이므로 십진수이며, 여기서 십이라는 것을 기수라 한다.

시스템 내부에서는 십진수가 아닌 이진수를 사용하여 저장한다. 디지털 신호에서 전기가 흐를 경우 참을 의미하는 1, 흐르지 않을 경우 거짓의 0 으로 표현되므로, 컴퓨터 내부에서 처리하는 숫자는 0 과 1 을 표현하는 이진수 체계를 사용한다.

이진수는 수의 자릿수에 사용할 수 있는 숫자가 0 과 1, 2 개 이므로 이진수라 한다.

2) 정보의 표현, 비트와 바이트

정보 처리 단위 중에서 가장 작은 기본 정보 단위가 비트(bit)이다. 비트가 연속적으로 8 개 모인 정보 단위를 바이트(byte)라 한다. 바이트가 4 개, 8 개 모이면 워드(word)라 한다.

3) 논리와 문자 표현

참과 거짓을 의미하는 두 가지 정보를 논리값이라 한다. 부울 대수는 컴퓨터가 정보를 처리하는 방식에 대하여 이론적인 배경을 제공하며, 0 과 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 둘 중 하나만 1 이면 결과가 1 이다. 항이 하나인 NOT 연산은 항이 0 이면 1 로, 1 이면 0 인 결과를 반환하는 연산이다.

아스키 코드는 1967 년에 표준으로 제정되어 1986 년에 마지막으로 개정되었다. 아스키코드를 기반으로 하고 있는 장치들 간에는 이 표준에 의해서 서로 문자를 주고 받을 수 있다.

유니코드는 전 세계 모든 언어를 하나의 코드 체계 안으로 통합하기 위하여 만들어진 코드이다. 유니코드는 전 세계의 모든 문자를 컴퓨터에서 일관되게 표현하고 다룰 수 있도록 설계된 산업 표준이며, 유니코드 협회가 제정하여 1991 년 버전 1.0 이 발표되었다.

1.5 소프트웨어 개발

1) 소프트웨어와 알고리즘

소프트웨어는 보통 프로그램이라고 부르는 것 외에도 데이터와 문서까지를 포함하는 포괄적인 개념이다.

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

2) 소프트웨어 개발과정

① 요구사항 또는 해결할 문제를 먼저 파악한다

② 프로그램을 설계한다

③ 설계된 알고리즘에 따라 코딩한다

④ 작성된 프로그램을 테스트 한다

⑤ 프로그램을 문서화하고 유지 보수한다

1.6 다양한 프로그래밍 언어

1) 50~60 년대에 개발된 프로그래밍 언어

- 포트란

  - IBM 704 시스템에서 과학과 공학 및 수학적 문제들을 해결하기 위해 고안된 프로그래밍 언어로 널리 보급된 최초의 고급 언어

- 코볼

  - 협회 CODASYL 이 1960 년 개발한 사무처리에 적합한 프로그래밍 언어

- 알골

  - 알고리즘을 표현하기 위한 언어를 줄여서 만든 이름으로, 포트란이 미국을 중심으로 사용했다면 알골은 유럽을 중심으로 과학기술 계산용 프로그래밍 언어로 사용되었다.

- 베이직

  - 초보자의 다목적용, 부호를 사용하는 명령어 코드의 약어로 초보자도 쉽게 배울 수 있도록 만들어진 대화형 프로그래밍 언어

2) 70 년대 이후 개발된 주요 프로그래밍 언어

- 파스칼

  - 프랑스의 수학자인 파스칼의 이름에서 따온 언어, 프로그래밍을 작성하는 방법인 알고리즘 학습에 적합하도록 개발된 프로그래밍 언어

- C++

  - 객체지향 프로그래밍을 지원하기 위해 C 언어가 가지는 장점을 그대로 계승하면서 객체의 상속성 등의 개념을 추가한 효과적인 언어

- 파이썬

  - 1991 년 네덜란드의 귀도 반 로섬이 개발한 객체지향 프로그래밍 언어

- 자바

  - 1995 년에 공식 발표되었으며 C++를 기반으로 한 객체지향 프로그래밍 언어

CHAPTER 02 C 프로그래밍 첫 걸음

2.1 프로그램 구현 과정

1) 프로그램 구상과 소스편집

프로그램을 구현하기 위해서는 프로그램 구상, 소스편집, 컴파일, 링크, 실행의 5 단계를 거친다.

소스 또는 소스코드는 선정된 프로그래밍 언어인 C 프로그램 자체로 만든 일련의 명령문을 의미한다. C 와 같은 프로그래밍 언어로 원하는 일련의 명령어가 저장된 파일을 소스파일이라 하며 일반 텍스트파일로 저장되어야 한다. 소스파일은 프로그래밍 언어에 따라 고유한 확장자를 갖는데, C 언어는 .c 이며 자바는 .java 이다.

2) 컴파일

컴파일러는 고급언어인 프로그래밍 언어로 작성된 소스파일에서 기계어로 작성된 목적파일을 만들어 내는 프로그램이다. 컴파일러에 의해 처리되기 전의 프로그램을 소스코드라면 컴파일러에 의해 기계어로 번역된 프로그램은 목적코드라 한다.

3) 링크와 실행

링커는 하나 이상의 목적파일을 하나의 실행파일로 만들어 주는 프로그램이다. 여러 개의 목적파일을 연결하고 참조하는 라이브러리를 포함시켜 하나의 실행파일을 생성하는데, 말 뜻 그대로 이 과정을 링크, 혹은 링킹이라 한다.

프로그램을 작성할 때, 프로그래머마다 새로 작성할 필요 없이 개발환경에서 미리 만들어 컴파일해 저장해 놓는데, 이 모듈을 라이브러리라 한다. 라이브러리란 공용으로 사용하기 위해 이미 만든 목적코드로 파일 .lib 또는 .dll 등으로 제공된다.

비주얼 스튜디오에서는 컴파일과 링크 과정을 하나로 합쳐 빌드(build)라 한다. 즉 빌드는 컴파일한 후, 계속해서 자동으로 링크를 수행하는 과정을 말 한다.

4) 오류와 디버깅

프로그램 개발 과정에서 나타나는 문제를 오류 혹은 에러(error)라고 한다. 오류는 그 발생 시점에 따라 컴파일(시간) 오류와 링크(시간) 오류, 실행(시간) 오류로 구분할 수 있다.

오류의 원인과 성격에 따라, 프로그래밍 언어 문법을 잘못 기술한 문법 오류와 내부 알고리즘이 잘못되거나 원하는 결과가 나오지 않은 등의 논리 오류로 분류할 수 있다.

프로그램 개발 과정에서 발생하는 다양한 오류를 찾아 소스를 수정하여 다시 컴파일, 링크, 실행하는 과정을 디버깅이라 하며 이를 도와주는 프로그램을 디버거라 한다.

2.2 C 프로그램의 이해와 디버깅 과정

1) 함수 개요와 시작함수 main()

C 프로그램과 같은 절차지향 프로그램은 함수로 구성된다. 프로그래머가 직접 만드는 함수를 사용자 정의 함수라 하며, 시스템이 미리 만들어 놓은 함수를 라이브러리 함수라 한다.

- 함수의 용어

  - 함수 정의: 사용자 정의 함수를 만드는 과정
  - 함수 호출: 라이브러리 함수를 포함해 만든 함수를 사용하는 것
  - 매개변수: 함수를 정의할 때 나열된 여러 입력 변수
  - 인자: 함수 호출 과정에서 전달되는 여러 입력값

main()은 사용자가 직접 만드는 함수 정의 과정이며, puts()는 함수 호출 문장이다. 프로그램이 실행되면 운영체제는 프로그램에서 가장 먼저 main()함수를 찾고 입력 형태의 인자 main() 함수를 호출한다. 호출된 main()함수의 첫 줄을 시작으로 마지막 줄까지 실행하면 프로그램은 종료된다. 이 함수를 CRT 시작함수라 하며 반드시 정의되어야 한다.

2) 여러 줄에 문자열을 출력

함수 puts()는 문자열을 전용으로 출력하는 함수이며, 함수 printf("")는 호출 시 전달되는 "문자열"과 같은 다양한 형태의 인자를 적절한 형식으로 출력하는 함수이다. 라이브러리 함수 puts()와 printf()를 사용하려면 첫 줄에 #include <stdio.h>를 넣어야 한다.

함수 puts()는 원하는 문자열을 괄호(" ")사이에 기술하면 그 인자를 현재 위치에 출력한 후 다음 줄 첫 열로 이동하여 출력을 기다리는 함수이다.

함수 printf()는 원하는 문자열을 괄호(" ")사이에 기술하면 그 인자를 현재 줄의 출력위치에 출력하는 함수이다.

CHAPTER 03 자료형과 변수

3.1 프로그래밍 기초

1) 키워드와 식별자

프로그래밍 언어에서는 문법적으로 고유한 의미를 갖는 예약된 단어가 있다. 이러한 예약어는 키워드라고도 한다. (auto, do, goto, break, if, int, long, void...)

프로그래머가 자기 마음대로 정의해서 사용하는 단어는 식별자이다. 하지만 이러한 식별자에도 사용 규칙들이 있다.

- C 프로그램에서의 예약어, 키워드와 비교하여 철자, 대문자, 소문자 등 무엇이라도 달라야 한다.
- 식별자는 영문자(대소문자 알파벳), 숫자(0~9), 밑줄(_)로 구성되며, 식별자의 첫 문자로 숫자가 나올 수 없다.
- 프로그램 내부의 일정한 영역에서는 서로 구별되어야 한다.
- 키워드는 식별자로 이용할 수 없다.
- 식별자는 대소문자를 모두 구별한다.
- 식별자 중간에 공백(space)문자가 들어갈 수 없다.

3.2 자료형과 변수선언

1) 자료형과 변수 개요

C 프로그래밍 언어에서 다루는 다양한 자료는 기본형, 유도형, 사용자정의형 등으로 나눌 수 있으며 기본형은 다시 정수형, 실수형, 문자형, void 로 나뉜다. 프로그래머는 이러한 자료에 적합한 알고리즘을 적용해 프로그램을 작성한다. 즉 자료형은 프로그래밍 언어에서 자료를 식별하는 종류를 말한다.

프로그래밍에서 정수와 실수, 문자 등의 자료값을 중간 중간에 저장할 공간이 필요하다. 이 저장 공간을 변수(variables)라 부르는데, 변수에는 고유한 이름이 붙여지며, 물리적으로 기억장치인 메모리에 위치한다. 변수는 선언된 자료형에 따라 변수의 저장공간 크기와 저장되는 자료값의 종류가 결정된다. 저장되는 값에 따라 변수값은 바뀔 수 있으며 마지막에 저장된 하나의 값만 저장 유지된다.

2) 변수선언과 초기화

변수선언은 컴파일러에게 프로그램에서 사용할 저장 공간인 변수를 알리는 역할이며, 프로그래머 자신에게도 선언한 변수를 사용하겠다는 약속의 의미가 있다. 변수는 고유한 이름이 붙여지고 자료값이 저장되는 영역이다. 프로그램에서 변수를 사용하려면 원칙적으로 사용 전에 변수선언 과정이 반드시 필요하다.

- 변수선언은 자료형을 지정한 후 고유한 이름인 변수이름을 나열하여 표시한다.
- 자료형은 int, double, float 와 같이 원하는 자료형 키워드를 사용하며, 변수 이름은 관습적으로 소문자를 이용하며, 사용 목적에 알맞은 이름으로 특정한 영역에서 중복되지 않게 붙이도록 한다.
- 변수선언도 하나의 문장이므로 세미콜론으로 종료된다.
- 변수선언 이후에는 정해진 변수이름으로 값을 저장하거나 참조할 수 있다.

  예) double height;

원하는 자료값을 선언된 변수에 저장하기 위해서는 대입연산자를 사용한다. 대입연산자 '='는 오른쪽에 위치한 값을 이미 선언된 왼쪽 변수에 저장한다는 의미이다.

예) int agel

age = 20;

age = 21;

변수를 선언만 하고 자료값을 아무것도 저장하지 않으면 원치 않는 값이 저장되며, 오류가 발생한다. 그러므로 변수를 선언한 이후에는 반드시 값을 저장하도록 한다. 이를 변수의 초기화라 한다.

예) int year = 1972;

LAB

두 정수의 합과 두 실수의 차를 다음 결과 창으로 출력되는 프로그램을 작성한다.

- 정수를 위한 자료형은 int 로, 실수를 위한 자료형은 double 로 이용
- 합을 위한 연산자+, 두 실수의 차를 위한 연산자- 와 결과 저장을 위한 변수 difference

  [실행결과]

  합: 73

  차: -7.003000

```c
// basictype.c: 두 정수의 합, 두 실수의 차 출력

#include <stdio.h>

int main(void)
{
        int a = 30, b = 43; //두 정수 선언과 초기 값 대입
        int sum;                //두 정수의 합을 저장할 변수 선언
        sum = a + b;            //두 정수의 합 구하기
```

```
        double x = 38.342, y = 45.345;    //두 실수 선언과 초기 값 대입
        double difference;                                    //두 실수의 차을
                                          저장할 변수 선언
        difference = x - y;                                   //두 실수의 차 구하기

        printf("합: %d₩n", sum);          //두 정수의 합 출력
        printf("차: %f₩n", difference);   //두 실수의 차 출력

        return 0;
}
```

3.3 기본 자료형

1) 정수 자료형

정수형의 기본 키워드는 int 이다. int 형으로 선언된 변수에는 십진수, 팔진수, 십육진수의 정수가 다양하게 저장될 수 있다. short 는 int 보다 작거나 같고, long 은 int 보다 크거나 같다.

자료형 short 는 short int 라고도 하며, long 은 long int 라고도 한다. 즉 자료형 short int 와 long int 에서 int 는 생략 가능하다. 정수형 short, int, long 은 모두 양수, 0, 음수를 모두 표현할 수 있다. 그러므로 [부호가 있는]을 의미하는 signed 키워드는 정수형 자료형 키워드 앞에 표시될 수 있다. 물론 signed 키워드는 생략될 수 있다.

0 과 양수만을 처리하는 정수 자료형은 short, int, long 앞에 키워드 unsigned 를 표시한다. 즉 부호가 없는 정수인 unsigned int 는 0 과 양수만을 저장할 수 있는 정수 자료형이다.

음수지원 여부	자료형	크기	표현범위
부호가 있는 정수형 signed	signed short	2 바이트	-32,768 ~ -32,768
부호가 있는 정수형 signed	signed int	4 바이트	-2,147,483,648 ~ 2,147,483,648
부호가 있는 정수형 signed	signed long	4 바이트	-2,147,483,648 ~ 2,147,483,648
부호가 있는 정수형 signed	unsigned short	2 바이트	0 ~ 63,535
부호가 있는 정수형 signed	unsigned int	4 바이트	0 ~ 4,294,967,295
부호가 있는 정수형 signed	unsigned long	4 바이트	0 ~ 4,294,967,295,

다음은 자료형 long long 으로 지원 가능한 우주의 행성 간의 거리를 프로그래밍 한 소스이다.

```
/*
        솔루션 / 프로젝트 / 소스파일: Ch03 / Prj05 / integer.c
        정수형 자료형 변수의 선언과 활용
        V 1.0 2016.
*/

#include <stdio.h>

int main(void)
{
        short sVar = 32000;                     //-32767에서 32767까지
        int   iVar = -2140000000;//약 21억 정도까지 저장 가능

        unsigned short int      usVar = 65000;          //0에서 65535까지 저장 가능
        unsigned int            uiVar = 4280000000;     //약 0에서 42억 정도까지 저장 가능

        printf("저장 값: %d %d₩n", sVar, iVar);
        printf("저장 값: %u %u₩n", usVar, uiVar);

        long long dist1 = 2720000000000; //지구와 천왕성 간의 거리(km) 27억 2천
        __int64 dist2 = 4500000000000; //태양과 해왕성 간의 거리(km) 45억

        printf("지구와 천왕성 간의 거리(km): %lld₩n", dist1);
        printf("태양과 해왕성 간의 거리(km): %lld₩n", dist2);

        return 0;
}
```

[실행결과]

저장값: 32000 -2140000000

저장값: 65000 4280000000

지구와 천왕성 간의 거리(km): 2720000000000

태양과 해왕성 간의 거리(km): 4500000000000

2) 부동소수 자료형

부동소수형은 실수를 표현하는 자료형이다. 부동소수형을 나타내는 키워드는 float, double, long double 세 가지이다. float 형 변수에 저장하면 꼭 3.14F 와 같이 float 형 상수로 저장한다.

자료형	크기	정수의 유효자릿수
float	4 바이트	6 ~ 7
double	8 바이트	15 ~ 16
long double	8 바이트	15 ~ 16

3) 문자형 자료형

문자형 자료형은 char, signed char, unsigned char 세 가지 종류가 있다.

자료형	저장공간 크기	표현범위
char	1 바이트	-128 ~ 127
signed char	1 바이트	-128 ~ 127
unsigned char	1 바이트	0 ~ 255

4) 아스키 코드

C 언어에서 문자형 자료공간에 저장되는 값은 실제로 정수값이며, 이 정수는 아스키 코드표에 의한 값이다. 아스키 코드는 ANSI 에서 제정한 정보 교환용 표준 코드로 총 127 개의 문자로 구성된다.

LAB

아스키 코드값 126 번은 물결문자 '~'이다. 다음 정보를 이용하여 다음 결과 창으로 출력되는 프로그램을 작성한다.

- 문자 '~'의 코드값: 십진수 126, 팔진수 176, 십육진수 72
- 출력을 위한 함수 printf()에서 %d 로 정수를, %c 로 문자를 출력

[실행결과] 126

~

~

~

```
// intchar.c: 아스키 토드 값 126 문자 '~'의 다양한 출력

#include <stdio.h>

int main(void)
{
        int ch = 126;

        printf("%d₩n", ch); //십진 코드 값 출력
        printf("%c₩n", ch); //문자 출력
        printf("%c₩n", '₩176'); //문자 출력
        printf("%c₩n", '₩x7e'); //문자 출력

        return 0;
}
```

5) 자료형의 크기

연산자 sizeof 를 이용하면 자료형, 변수, 상수의 저장공간 크기를 바이트 단위로 알 수 있다.

예) sizeof(char) / sizeof 3.14 / sizeof n

자료형의 범주에서 벗어난 값을 저장하면 오버플로와 언더플로가 발생한다. 자료형 unsigned char 은 8 비트로 0 에서 255 까지 저장 가능하다. 마찬가지로 만일 256 을 저장하면 0 으로 저장된다.

정수형 자료형에서 최대값+1 은 오버플로로 인해 최소값이 된다. 마찬가지로 최소값-1 은 최대값이 된다. 이러한 특징을 정수의 순환이라고 한다.

실수형 float 변수에 1.1175E-50 과 같이 부동소수점수가 너무 많아 정밀도가 매우 자세한 수를 저장하면 언더플로가 발생하여 0 이 저장된다.

3.4 상수 표현방법

1) 상수의 개념과 표현방법

상수(constant)는 이름 없이 있는 그대로 표현한 자료값이나 이름이 있으나 정해진 하나의 값만으로 사용되는 자료값을 말한다. 상수는 크게 분리하면 리터럴 상수와 심볼릭 상수로 구분될 수 있다.

리터럴 상수는 달리 이름이 없어 소스에 그대로 표현해 의미가 전달되는 다양한 자료값을 말한다. 즉 10, 23.4 과 같은 수 , "c 는흥미로워요"와 같은 문자열이 그 예이다. 심볼릭 상수는 리터럴 상수와 다르게 변수처럼 이름을 갖는 상수를 말한다. 심볼릭 상수를 표현하는 방법은 const 상수, 매크로 상수, 그리고 열거형 상수를 이용하는 세가지 방법이 있다.

구분	표현 방법	설명	예
리터럴 상수 (이름이 없는 상수)	정수형, 실수형, 문자, 문자열 상수	다양한 상수를 있는 그대로 기술	32, 025, '₩n', "C 언어", 3.2F
심볼릭 상수 (이름이 있는 상수)	const 상수	키워드 const 를 이용한 변수 선언과 같으며, 수정할 수 없는 변수 이름으로 상수 정의	const double PI = 3.141592;
	매크로 상수	전처리가 명령어 #define 으로 다양한 형태를 정의	#define PI = 3.141592;
	열거형 상수	정수 상수 목록 정의	enum bool {false,true}

2) 정수와 실수 리터럴 상수

정수형 상수는 int, unsigned, long 등의 자료형으로 나뉜다. 일반적으로 상수의 정수표현은 십진수로 인식되나, 숫자 0 을 정수 앞에 놓으면 팔진수로 인식한다. 그러므로 0 뒤에 수는 숫자 0 에서 7 까지 만으로 구성되어야 한다. 숫자 0 과 알파벳으로 0x 또는 0X 를 숫자 앞에 놓으면 십육진수로 인식한다.

- 팔진수와 십육진수의 상수 표현

  - 숫자 앞에 0 표시: 팔진수 (06, 020, 030)

  - 숫자 앞에 영과 문자 x 조합인 0x 또는 0X 앞에 표시: 십육진수 (0Xa3, 0x5D)

실수는 e 또는 E 를 사용하여 10 의 지수표현 방식으로 나타낼 수 있다. 형식 제어문자 %f 로 출력되는 실수는 소수점 6 자리까지 출력된다.

예시) printf("%f", 3.14E+2);

실수형 상수도 float, double, long double 의 자료형으로 나뉜다. 즉 소수는 double 유형이며, float 상수는 숫자 뒤에 f 나 F 를 붙인다.

- float : f , F - long double: l, L

```c
/*
        솔루션 / 프로젝트 / 소스파일: Ch03 / Prj11 / numliterals.c
        정수형 실수형 리터럴 상수의 다양한 표현
        V 1.0 2016.
*/

#include <stdio.h>

int main(void)
{
        printf("%d ", 30);              printf("%d ", 10);
        //십진수
        printf("%d ", 030);             printf("%d ", 010);
        //팔진수
        printf("%d ", 0X2F);    printf("%d₩n", 0x1b);    //십육진수

        printf("%f ", 3.14);    printf("%f ", 2.0);
        printf("%f ", 3.14E+2); printf("%f ", 21.8e2);
        printf("%f ", 3.14E-2); printf("%f₩n", 218e-3);

        return 0;
}
```

3) 심볼릭 const 상수

변수선언 시 자료형 또는 변수 앞에 키워드 const 가 놓이면 이 변수는 심볼릭 상수가 된다. 심볼릭 상수는 일반 변수와는 달리 초기값을 수정할 수 없으며, 이름이 있는 심볼릭 상수가 된다. 상수는 변수선언 시 반드시 초기값을 저장하고 대문자로 선언해야 한다.

- 키워드 const 의 위치

  (const) double (const) RATE = 0.03 ;

  // 키워드 const 의 위치는 자료형과 변수 앞 둘 다 가능하다.

4) 열거형 상수

열거형은 키워드 enum 을 사용하여 정수형 상수 목록 집합을 정의하는 자료형이다. 열거형 enum 을 사용하여 열거형 정수 상수를 한번에 여러 개 정의하여 활용할 수 있다. 즉 열거형 상수에서 목록 첫 상수의 기본값이 0 이며 다음부터 1 씩 증가하는 방식으로 상수값이 자동으로 부여된다.

- 열거형 상수 목록 선언

  enum DAY {SUN, MON, TUE, WED, THU, FRI, SAT} ;

  enum 열거형태그명 {열거형상수 1, 열거형상수 2, 열거형상수 3,...} ;

다음과 같이 상수 목록에 특정한 정수값을 부분적으로 지정할 수도 있다. 상수값을 지정한 상수는 그 값으로, 따로 지정되지 않은 첫 번째 상수는 0 이며, 중간 상수는 앞의 상수보다 1 씩 증가한 상수값으로 정의된다.

- 직접 정수를 지정하는 열거형 상수 목록

  enum SHAPE = {POINT, LINE, TRI=3, RECT, OCTA=8,CIRCLE} ;

5) 매크로 상수

전처리 지시자 #define 은 매크로 상수를 정의하는 지시자이다. 다른 일반 변수와 구분하기 위해 #define 에 의한 심볼릭 상수도 주로 대문자 이름으로 정의하는데, 이를 매크로 상수라고 부른다.

- 매크로 상수 KPOP 과 PI

  #define KPOP 50000 //정수 매크로 상수

  #define PI 3.14 //실수 매크로 상수

문자형과 정수형의 최대 최소 상수는 헤더파일 limits.h 에 정의되어 있으며 부동소수형의 최대 최소 상수는 헤더파일 float.h 에 정의되어 있다.

CHAPTER 04 전처리와 입출력

4.1 전처리

1) 전처리 개요

C 언어는 컴파일러가 컴파일하기 전에 전처리기의 전처리 과정이 필요하다.

- 전처리 과정에서 처리되는 문장을 전처리 지시자라 한다.
- #include, #define 같은 전처리 지시자는 항상 #로 시작하고, 마지막에 세미콜론이 없는 등 일반 C 언어 문장과는 구별된다.

대표적인 헤더파일인 <stdio.h>는 printf(), puchar(), scanf() 등과 같은 입출력 함수를 위한 함수원형 등이 정의된 헤더파일이다. 즉 위와 같은 입출력 함수를 사용하는 소스에서는 헤더파일 <stdio.h>가 반드시 필요하다.

전처리 지시자 #include 는 헤더파일을 삽입하는 지시자이다. #include <stdio.h>는 명시된 헤더파일 stdio.h 를 그 위치에 삽입하는 역할을 수행한다.

2) 전처리 지시자 #define

전처리 지시자 #define 은 매크로 상수를 정의하는 지시자이다. #define 에 의한 심볼릭 상수도 주로 대문자 이름으로 정의하는데, 이를 매크로 상수라고 부른다.

지시자 #define 에서 기호상수 뒤에 오는 치환문자열에는 일반 상수, 상수의 연산, 이미 정의 된 기호상수 등이 올 수 있다. #define 에서 그 활용도를 높이기 위한 방안이 함수와 같이 인자를 이용하는 방법이다. 기호 상수에서 이름 뒤의 괄호 사이에 인자를 이용할 수 있다.

- 인자가 있는 매크로의 치환

  #define SQUARE(x)　　((x) * (x))

  printf("%d, %d", SQUARE(2), SQUARE(3)) ;

  //　= printf("%d, %d", SQUARE(2*2), SQUARE(3*3)) ;

그러나 괄호를 생략하면 오류가 발생하기 때문에 매크로를 구성하는 모든 인자와 외부에 괄호를 이용해야 한다. 또한 매크로는 이미 정의된 매크로를 다시 사용할 수 있다.

- 인자를 사용하는 다양한 매크로

  #define SQUARE (x)　　((x)*(x))

  #define CUBE (x)　　(SQUARE(x) * (x))

LAB

매크로 myprint(x)를 정의해 인자인 문자열 x 를 한 행에 출력하는 프로그램

- 전처리 지시자 #define 으로 인자가 있는 매크로 myprint(x)를 정의

  [실행결과]

  매크로로 출력하기

  출력함수로 출력하기

```
// file: basicmacro.h

#include <stdio.h>

#define myprint(x) printf(x); ₩
                                puts("")

int main(void)
{
        myprint("매크로로 출력하기");
        printf("출력함수로 출력하기₩n");

        return 0;
}
```

4.2 출력 함수 printf()

1) 출력함수 printf()에서의 형식지정자의 이해

printf()의 인자는 크게 형식문자열과 출력할 목록으로 구분된다. 출력 목록의 각 항목을 형식문자열에서 %d 와 같이 %로 시작하는 형식지정자 순서대로 서식화하여 그 위치에 출력한다.

- 출력 함수 printf() 개요

  int term = 15;

  printf("%d 의 두 배는 %d 입니다.", term, 2*term) ;

함수 printf()의 첫 번째 인자인 형식문자열은 일반문자와 이스케이프 문자, 그리고 형식 지정자로 구성된다. 결론적으로 출력하는 문자열 내부에서 적당한 정수와 실수 문자열 등의 자료형을 적당한 위치에 출력하기 위해 형식 지정자와 이스케이프 문자 등이 필요한 것이다. 형식 지정자는 출력 내용의 자료형에 따라 %d, %i, %c, %s 와 같이 %로 시작한다.

2) 정수의 십진수, 팔진수, 16 진수의 출력

- 정수의 십진수 출력을 위하 형식 지정자는 %d 와 %i 이다.
- 정수를 8 진수로 출력하려면 %o 를 이용하며, 앞 부분에 숫자 0 이 붙는 출력을 하려면 %#o 을 이용한다.
- 정수를 소문자의 십육진수로 출력하려면 %x 와 대문자로 출력하려면 %X 를 이용하며, 출력되는 16 진수 앞에 0x 또는 0X 를 붙여 출력하려면 #를 삽입하여 %#x 를 이용한다.

함수 printf()는 두 번째 인자부터 시작되는 인자의 값을 형식 지정자에 맞게 서식화하여 출력하며, 반환값은 출력한 문자 수이며 오류가 발생하면 음수를 반환한다.

3) 실수를 위한 출력

- 실수의 간단한 출력을 위한 형식 지정자는 %i
- 형식 지정자 %f 는 실수를 기본적으로 소수점 6 자리까지 출력한다.
- 함수 printf()에서 실수 출력으로 %f 와 함께 %lf 도 사용된다.

4) 출력 폭의 지정

출력 필드 폭이 출력 내용의 폭보다 넓으면 정렬은 기본이 오른쪽이며, 필요하면 왼쪽으로 지정할 수 있다.

- 형식지정자 %8d 는 십진수를 8 자리 폭에 출력한다. 지정한 출력 폭이 출력할 내용보다 넓으면 정렬은 기본적으로 오른쪽이다.
- 출력 폭을 지정하며 정렬을 오른쪽으로 지정하려면 %-8d 처럼 앞에 -를 삽입한다.

부동소수에서 소수점 이하 자릿수를 지정하려면 %[전체폭].[소수점이하폭]f 와 같이 표시한다.

- 지정된 폭이 출력값의 폭보다 넓으면 기본으로 오른쪽 정렬을 하며, 왼쪽 정렬을 하려면 %-10.3f 와 같이 폭 앞에 -를 넣는다.
- %+10.3f 와 같이 폭 앞에 +를 넣으면 양수라도 +부호가 표시된다.

5) 형식 지정자 개요

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.dddddd Exxx의 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

표 4-4 옵션지정 문자(flags) 종류

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

형식지정자	정수	결과
%d	1234	1234
%6i	1234	__1234
%+6i	1234	+_1234
%-06i	1234	+01234
%60	037	____37
%-60	037	37

형식지정자	정수	결과
%-#6o	037	037___
%#-6o	037	037___
%05x	0x1f	0001f
%0#6x	0x1f	0x001f
%-0#5x	0x1f	0x1f_
%#6X	0x1f	__0X1F

LAB

정수와 실수, 문자와 문자열의 출력

- 자료형 int 형인 나이와 double 형 성적 평균평점 등을 출력
- 성별, 나이, 몸무게, 평균평점을 다음과 같이 출력

  [실행결과]

  성별: M

  이름: 안 병훈

  나이: 20

  몸무게: 62.49

  평균평점(GPA): 3.880

```c
// basicoutput.c:

#include <stdio.h>

int main(void)
{
        int age = 20;
        double gpa = 3.88f;
        char gender = 'M';
        float weight = 62.489F;

        printf("성별: %c₩n", gender);
        printf("이름: %s₩n", "안 병훈");
        printf("나이: %d₩n", age);
        printf("몸무게: %.2f₩n", weight);
        printf("평균평점(GPA): %.3f₩n", gpa);

        return 0;
}
```

4.3 입력 함수 scanf()

1) 함수 scanf()

scanf()는 대표적인 입력함수이다. 함수 printf()함수처럼 함수 scanf()에서 첫 번째 인자는 형식 문자열이라 하며 형직지정자와 일반문자로 구성된다. 형식 지정자는 %d, %c 와 같이 %로 시작한다. 함수 scanf()에서 두 번째 인자부터는 입력 변수 목록으로 변수 이름 앞에 반드시 주소연산자 &를 붙여 나열한다.

- &는 주소연산자로 뒤에 표시된 피연산자인 변수 주소값이 연산값으로, scanf()의 입력변수 목록에는 키보드에 입력값이 저장되는 변수를 찾는다는 의미에서 반드시 변수의 주소연산식 &가 사용되어야 한다.
- 만일 주소연산이 아닌 변수로만 기술하면 오류가 발생한다.

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

#include <stdio.h>

int main(void)
{
        int snum, credit;

        printf("당신의 학번과 신청학점은? ");
        scanf("%d%d", &snum, &credit);
        printf("학번: %d 신청학점: %d₩n", snum, credit);

        return 0;
}
```

[실행결과]

당신의 학번과 신청학점은? 20164108 22

학번: 20164108 신청학점: 18

2) 함수 getchar()와 putchar()

함수 getchar()은 영문 'get character'의 의미로 문자 하나를 입력하는 매크로 함수이고, putchar()은 반대로 출력하기 위한 매크로 함수이다. 이 함수를 이용하려면 printf()나 scanf()처럼 헤더파일 stdio.h 가 필요하다.

함수 getchar()는 인자 없이 함수를 호출하며 입력된 문자값을 자료형 char 나 정수형으로 선언된 변수에 저장할 수 있다.

```
char a = getchar() ;
```

함수호출 putchar('a')는 인자인 'a'를 출력하는 함수로 사용된다.

```c
// file: putchar.c

#include <stdio.h>

int main(void)
{
        char a = '₩0';

        puts("문자 하나 입력:");
        a = getchar();
        putchar(a); putchar('₩n');

        return 0;
}
```

[실행결과]

문자 하나 입력:

#

#

CHAPTER 05 연산자

5.1 연산식과 다양한 연산자

1) 연산식과 연산자 분류

변수와 다양한 리터럴 상수 그리고 함수의 호출 등으로 구성되는 식을 연산식이라 한다. 연산식은 반드시 하나의 결과값인 연산값을 갖는다. 연산자는 산술연산자 +,-,* 기호와 같이 이미 정의된 연산을 수행하는 문자 또는 문자조합 기호를 말한다. 그리고 연산에 참여하는 변수나 상수를 피연산자라 한다.

연산식은 평가하여 항상 하나의 결과값을 갖는다. 연산식의 결과값은 간단히 연산값이다.

피연산자의 개수에 따라 단항, 이항, 삼항 연산자로 나눌 수 있다. 삼항 연산자는 조건연산자 ?:가 유일하다. ++a 처럼 연산자가 앞에 있으면 전위 연산자이며 a++와 같이 연산자가 뒤에 있으면 후위 연산자라고 한다.

2) 산술연산자와 부호연산자

산술연산자는 +, -, *, /, %로 더하기, 빼기, 곱하기, 나누기, 나머지 연산자이다. 정수끼리의 나누기 연산결과는 소수 부분을 버린 정수이다. 나머지 연산식 a % b 의 결과는 a 를 b 로 나눈 나머지 값이다.

3) 대입 연산자와 증감 연산자

대입연산자는 =으로 연산자 오른쪽의 연산값을 변수에 저장하는 연산자이다. 대입연산자의 왼쪽 부분에는 반드시 하나의 변수만이 올 수 있다. 대입 연산식 a = a+b 는 중복된 a 를 생략하고 간결하게 a+=b 로 쓸 수 있다. 이와 같이 산술 연산자와 대입 연산자를 이어 붙인 연산자 +=, -=, /= 등을 축약 대입연산자라 한다.

증가연산자 ++와 감소연산자 –는 변수값을 각각 1 증가시키고, 1 감소시키는 기능을 수행한다. 증가연산자에서 n++와 같이 연산자 ++가 피연산자보다 뒤에 위치하는 후위이면 1 증가되기 전의 값이 연산 결과값이다. 반대로 ++n 과 같이 저누이이면 1 증가된 값이 연산 결과값이다.

증감연산자는 변수만을 피연산자로 사용할 수 있으며 상수나 일반 수식을 피연산자로 사용할 수 없다.

```c
// file: increment.c

#include <stdio.h>

int main(void)
{
        int m = 10, n = 5;
        int result;

        result = m++ + --n;
        printf("m=%d n=%d result=%d₩n", m, n, result);

        result = ++m - n--;
        printf("m=%d n=%d result=%d₩n", m, n, result);

        return 0;
}
```

[실행결과]

m=11 n=4 result=14

m=12 n=3 result=8

5.2 관계와 논리, 조건과 비트연산자

1) 관계와 논리연산자

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

- 관계연산자의 종류와 사용

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 큰거나 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)

논리연산자 &&, ||, !은 각각 and, or, not 의 논리연산을 의미하며 그 결과가 참이면 1, 거짓이면 0 을 반환한다. c 언어에서 참과 거짓의 논리형은 따로 없으므로 0 은 거짓을 의미하며 0 이 아닌 모든 정수와 실수는 모두 참을 의미한다.

x	y	x && y	x || y	!x
0(거짓)	0(거짓)	0	0	1
0(거짓)	Ø(0이 아닌 값)	0	1	1
Ø(0이 아닌 값)	0(거짓)	0	1	0
Ø(0이 아닌 값)	Ø(0이 아닌 값)	1	1	0

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

그림 5-13 논리연산자의 연산 결과

2) 조건 연산자

조건 연산자는 조건에 따라 주어진 피연산자가 결과값이 되는 삼항 연산자이다. 즉 연산식 (x ? a : b)에서 피연산자는 x, a, b 세 개이며 첫번째 피연산자인 x 가 참이면 결과는 a, x 가 0 이면 결과는 b 이다.

- 조건연산자 계산 방법

  x ? a : b

3) 비트 연산자

비트 논리 연산자는 피연산자 정수값을 비트 단위로 논리 연산을 수행하는 연산자로 피연산자가 두 개인 이항 연산자이다. 비트 연산은 각 피연산자를 int 형으로 반환하여 연산하며 결과도 int 형이다.

AND 연산자인 &는 두 비트 모두 1 이어야 1 이며, OR 연산자인 |는 하나만 1 이어도 1 이고, ^는 서로 다르면 1 이고, 같으면 0 이다. NOT 또는 보수 연산자인 ~은 단항 연산자로 0 인 비트는 1 로, 1 인 비트는 0 으로 모두 바꾸는 연산자이다.

표 5-4 각 비트 연산 방법

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

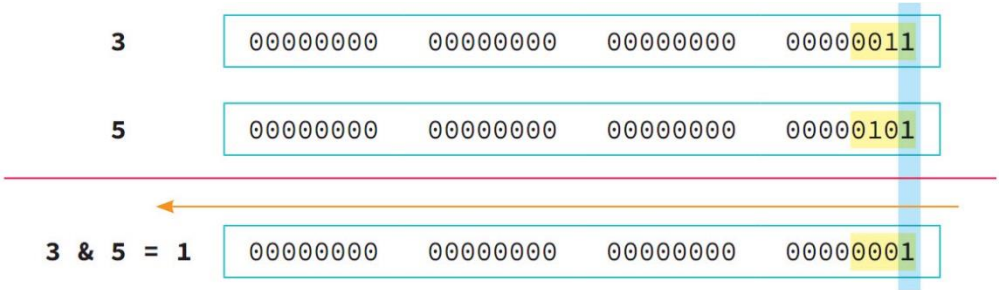


그림 5-17 비트 논리 연산 3 & 5

표 5-5 보수 연산 예

피연산자		보수 연산	
수	비트표현(이진수)	보수 연산 결과	십진수
1	000000000 000000000 000000000 000000001	11111111 11111111 11111111 11111110	~1 = -2
4	000000000 000000000 000000000 000000100	11111111 11111111 11111111 11111011	~4 = -5

표 5-6 다양한 비트 논리연산식

연산식	설명	연산값	연산식	설명	연산값
1 & 2	0001 & 0010	0	1 & 3	0001 & 0011	1
3 | 4	0011 | 0100	7	3 & 5	0011 | 0101	7
3 ^ 4	0011 ^ 0100	7	3 ^ 5	0011 | 0101	6
~2	-2 == ~2 + 1	-3	~3	-3 == ~3 + 1	-4

비트 이동 연산자 >>, <<는 연산자의 방향인 왼쪽이나 오른쪽으로, 비트 단위로 줄줄이 이동시키는 연산자이다. 비트 연산 >>와 <<은 오른쪽과 왼쪽에 빈 자리가 생겨, 오른쪽 빈자리 LSB 는 모두 0 으로 채워지며, 왼쪽 빈 자리 MSB 는 원래의 부호비트에 따라 0 또는 1 이 채워진다.

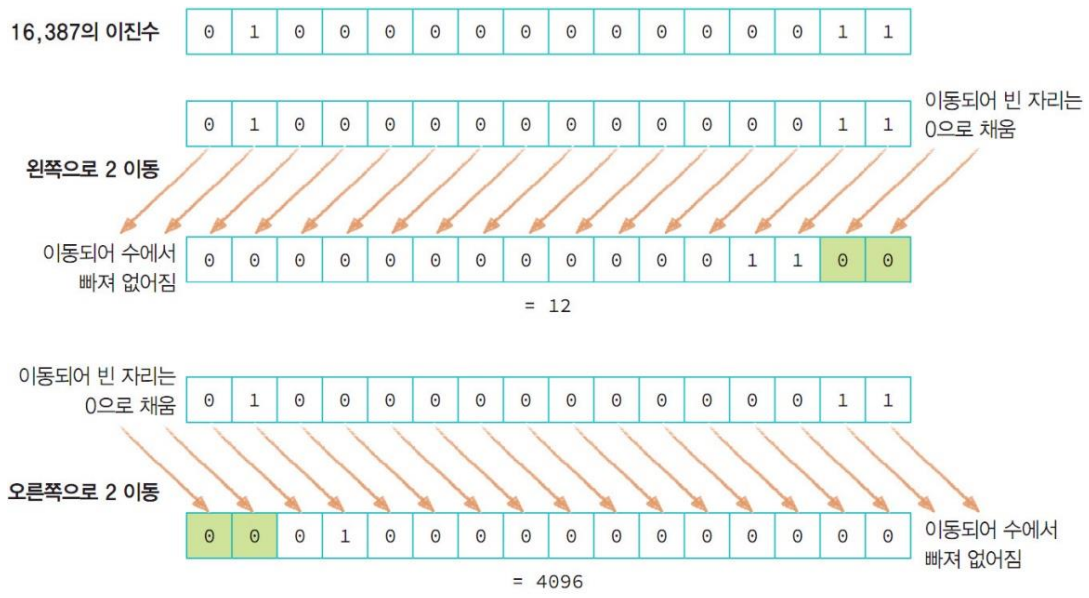


그림 5-20 비트 논리 연산 16387 << 2와 16387 >> 2

표 5-7 비트 이동 연산자

연산자	이름	사용	연산 방법	새로 채워지는 비트
>>	right shift	op1 >> op2	op1을 오른쪽으로 op2 비트만큼 이동	가장 왼쪽 비트인 부호 비트는 원래의 부호 비트로 채움
<<	left shift	op1 << op2	op1을 왼쪽으로 op2 비트만큼 이동	가장 오른쪽 비트를 모두 0으로 채움

표 5-8 다양한 비트 이동연산식

연산식	설명	연산값
15 << 1	0000 1111 << 1 0001 1110	30
0x30000000 << 2	최상위 4비트: 0011 0… << 2 1100 0…	-1073741824 0XC0000000
-30 >> 1	짝수이면 2로 나누기	-15

연산식	설명	연산값
15 << 2	0000 1111 << 2 0011 1100	60
-30 << 2	음수로 4배 커짐	-120
-30 >> 2	한 비트 이동마다. 음수 홀수는 -(a+1)한 수를 2로 나누기	-8

5.3 형변환 연산자와 연산자 우선순위

1) 내림변환과 올림변환

올림변환은 작은 범주의 자료형(int)에서 보다 큰 범주인 형(double)로의 형변환 방식이며, 내림변환은 큰 범주의 자료형 double 에서 보다 작은 범주인 int 형으로의 형변환 방식이다.

이러한 올림변환은 정보의 손실이 없으므로 컴파일러에 의해 자동으로 수행될 수 있다. 컴파일러가 자동으로 수행하는 형변환을 묵시적 형변환이라 한다.

컴파일러가 스스로 시행하는 묵시적 내림변환의 경우 정보의 손실이 일어날 수 있으므로 경고를 발생한다.

2) 형변환 연산자

형변환 연산자는 뒤에 나오는 피연산자의 값을 괄호에서 지정한 자료형으로 변환하는 연산자이다. 형변환 연산자를 사용한 방식을 명시적 형변환이라고 한다.

상수나 변수의 정수값을 실수로 변환하려면 올림변환을 사용한다. 반대로 실수의 소수부분을 없애고 정수로 사용하려면 내림변환을 사용할 수 있다.

(int) 30.535

double result = ((double) 7 / 2) ;

3) 연산자 sizeof 와 콤마연산자

연산자 sizeof 는 연산값 또는 자료형의 저장장소의 크기를 구하는 연산자이다. 연산자 sizeof 의 결과값은 바이트 단위의 정수이다. 연산자 sizeof 는 피연산자가 int 와 같은 자료형인 경우 반드시 괄호를 사용해야 한다.

- sizeof (int) / sizeof (3.14) / sizeof a

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

- 3 + 4, 5 – 10 //결과는 -5

4) 복잡한 표현식의 계산

- 첫 번째 규칙은 괄호가 있으면 먼저 계산하며
- 두 번째 규칙으로 연산의 우선순위이며,
- 세 번째 규칙은 동일한 우선순위인 경우 연산을 결합하는 방법인 결합성이다.

표 5-9 C 언어의 연산자 우선순위

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

표 5-10 연산 우선순위 예

변수값	표현식	설명	해석	결과
x = 3 y = 3	x >> 1 + 1 > 1 & y	산술 > 이동 > 관계 > 비트	((x >> (1 + 1)) > 1) & y	0
	x - 3 || y & 2	산술 > 비트 > 논리	(x - 3) || (y & 2)	1
	x & y && y >= 4	관계 > 비트 > 논리	(x & y) && (y >= 4)	0
	x && x | y++	증가 > 비트 > 논리	x && (x | (y++))	1

CHAPTER 06 조건

6.1 제어문 개요

1) 제어문의 종류

C 언어에서 제공되는 제어문은 조건선택과 반복(순환), 분기처리로 나눌 수 있다. 이러한 조건선택, 반복, 분기처리 구문을 이용하여 문장의 실행 순서를 다양화 시킬 수 있다.

조건선택 구문이란 두 개 또는 여러 개 중에서 한 개를 선택하도록 지원하는 구문이다.

- 조건선택 (조건에 대한 선택 구문)
  - if
  - if else
  - if else if
  - nested if
  - switch

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

- 반복 순환 (반복 조건에 따라 일정영역의 반복 구문)
  - For
  - While
  - Do while

분기 구문은 작업을 수행 도중 조건에 따라 반복이나 선택을 빠져 나가거나(break), 일정 구문을 실행하지 않고 다음 반복을 실행하거나(continue), 지정된 위치로 이동하거나(goto) 또는 작업 수행을 마치고 이전 위치로 돌아가는(return) 구문이다.

- 분기처리 (지정된 영역으로 실행을 이동하는 구문)
  - break
  - continue
  - goto
  - return

6.2 조건에 따른 선택 if 문

1) 조건에 따른 선택 개요

조건 선택의 예	기준변수	조건 표현의 의사코드
온도가 32 도 이상이면 폭염 주의를 출력	온도 temperature	if (temperature >=32) printf("폭염주의");
낮은 혈압이 100 이상이면 '고혈압 초기'로 진단	혈압 low_pressure	if (low_pressure >=100) printf("고혈압 초기");
속도가 40km 와 60km 사이면 '적정 속도' 출력	속도 speed	if (40 <= speed && speed <= 60) printf("적정속도");
면허시험에서 60 점 이상이면 합격, 아니면 불합격 출력	성적 point	if (point >=60) printf("합격"); else printf("불합격");

2) if 문장

문장 **if** 는 조건에 따른 선택을 지원하는 구문이다. 가장 간단한 if 문의 형태는 **if (cond) stmt;** 이다. if 문에서 조건식 cond 가 0 이 아니면(참) stmt 를 실행하고, 0 이면(거짓) stmt 를 실행하지 않는다.

문장 stmt 는 여러 문장이라면 블록으로 구성될 수 있으며, if 문이 종료되면 그 다음 문장이 실행된다. 문장 if 의 조건식(cond)는 반드시 괄호가 필요하며, 참이면 실행되는 문장은 반드시 들여쓰기를 한다.

예시) 학점이 3.2 이상이면 회사에 지원할 수 있고 축하메시지 입력

```
if (grade > =3.2)
    {
       printf("회사에 지원할 수 있습니다.₩n");
    }
    printf("졸업을 축하합니다.₩n");
```

- 잘못된 if (조건식); 과 if 조건식

```
- if ( grade > = 3.2 );          //학점이 3.2 미만이라도 다음 두문장은 항상 실행
      printf("회사에 지원할 수 있습니다.");
   print(" 졸업을 축하합니다.");
- if   grade >= 3.2      //if 문에서 조건식에는 괄호가 필요하다.
      printf("회사에 지원할 수 있습니다.");
   print(" 졸업을 축하합니다.");
```

다음은 표준입력으로 받은 온도가 32 도 이상이면 "폭염 주의보를 발령합니다" 와 "건강에 유의하세요."를 출력하며, 온도와 상관없이 항상 현재 온도를 출력하는 프로그램이다.

```c
// file: if.c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int main(void)
{
        double temperature;

        printf("현재 온도 입력: ");
        scanf("%lf", &temperature);

        if (temperature >= 32.0)
        {
                printf("폭염 주의보를 발령합니다.₩n");
                printf("건강에 유의하세요.₩n");
        }
        printf("현재 온도는 섭씨 %.2f 입니다.₩n", temperature);

        return 0;
}
```

[실행결과]

현재 온도 입력:31.3

현재 온도는 섭씨 31.30 입니다.

[실행결과]

현재 온도 입력:34.678

폭염 주의보를 발령합니다.

건강에 유의하세요.

현재 온도는 섭씨 34.68 입니다.

3) if else 문장

if 문은 조건이 만족되면 특정한 문장을 실행하는 구문이다. 반대로 조건이 만족되지 않은 경우에 실행할 문장이 있다면 else 를 사용한다.

조건문 if (cond) stmt; else stmt2; 는 조건 cond 를 만족하면 stmt1 을 실행하고, 조건 cond 를 만족하지 않으면 stmt2 를 실행하는 문장이다.

정수 n 이 짝수인지 홀수인지 판단할 수 있는 조건식으로 (n%2==0) 또는 (n%2)이 주로 사용된다.

예시) if (n % 2 == 0)

printf(“짝수”);

else

print(“홀수”);

- 조건문 if else 에서 주의해야 할 점

  - (조건식)은 괄호가 필요하다.

  - 조건식에서 등호를 대입으로 잘못 쓰는 것에 주의가 필요하다. 즉 (n==100) 을 (n=100) 로 잘못 쓰면 항상 참으로 인식한다.

  - if(조건식); 이나 else;와 같이 필요 없는 곳에 세미콜론을 넣지 않는다.

  - 조건식이 참이면 실행되는 stmt1 이나 거짓이면 실행되는 stmt2 부분이 여러 문장이면 {여러 문장}의 블록으로 구성한다.

다음은 표준입력으로 받은 정수의 짝수와 홀수를 판별하는 프로그램이다.

```c
// file: ifelse.c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int main(void)
{
	int n;

	printf("정수 입력: ");
	scanf("%d", &n);

	if (n % 2) // if (n % 2 != 0)
		printf("홀수");
	else
		printf("짝수");

	printf("입니다.₩n");

	//조건연산자 이용
	(n % 2) ? printf("홀수") : printf("짝수");
	printf("입니다.₩n");

	return 0;
}
```

[실행결과]

정수입력: 5

홀수입니다.

홀수입니다.

[실행결과]

정수입력: 6

짝수입니다.

짝수입니다.

다음은 입력된 정수의 3 의 배수를 판별하는 프로그램으로, 조건식이 참이거나 거짓인 경우, 블록을 사용하여 두 문장을 구현하였다.

```c
// file: multipleof3.c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int main(void)
{
	int n;

	printf("정수 입력: ");
	scanf("%d", &n);

	if (n % 3) // if (n % 3 != 0)
	{
		printf("입력된 %d는 3의 배수가 아닙니다.₩n", n);
	}
	else
	{
		printf("입력된 %d는 3의 배수입니다.₩n", n);
	}
	printf("조건식 %d %% 3의 결과는 %d입니다.₩n", n, n % 3);

	return 0;
}
```

[실행결과]

정수 입력: 3

입력된 3 은 3 의 배수입니다.

조건식 3 % 3 의 결과는 0 입니다.

[실행결과]

정수 입력: 4

입력된 4 은 3 의 배수가 아닙니다.

조건식 4 % 3 의 결과는 1 입니다.

cond1 조건식이 참이면 바로 아래의 문장 stmt1 를 실행하고 종료되며, 거짓이면 다음 조건식 cond2 로 계속 이어가며, 조건식이 모두 만족되지 않으면 결국 마지막 else 다음 문장 stmt4 를 실행한다. stmt1 에서 stmt4 에 이르는 여러 문장 중에서 실행되는 문장은 단 하나이다.

- 조건문 if else if

```
if (cond1)
        stmt1;
else if (cond2)
        stmt2;
else if (cond3)
        stmt3;
else
        stmt4;
next;
```

다음은 평균평점의 점수에 따라 다음과 같이 출력이 달라지는 예문이다.

```c
// file: ifelseif.c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int main(void)
{
        double gpa;

        printf("평균평점 입력: ");
        scanf("%lf", &gpa);

        if (gpa >= 4.3)
                printf("성적이 최고 우수한 학생입니다.₩n");
        else if (gpa >= 3.8)
                printf("성적이 매우 우수한 학생입니다.₩n");
        else if (gpa >= 3.0)
                printf("성적이 우수한 학생입니다.₩n");
        else
                printf("성적이 3.0 미만인 학생입니다.₩n");

        return 0;
}
```

[실행결과]

평균평점 입력: 4.3 성적이 최고 우수한 학생입니다.	평균평점 입력: 3.9 성적이 매우 우수한 학생입니다.
평균평점 입력: 3.3 성적이 우수한 학생입니다.	평균평점 입력: 2.7 성적이 3.0 미만인 학생입니다.

LAB

표준입력으로 받은 두 실수의 대소에 따라 다양한 연산을 수행하여 그 결과를 출력

- 만일 x > y 이면 x / y 연산값 출력
- 만일 x < y 이면 x + y 연산값 출력
- 만일 x == y 이면 x * y 연산값 출력

  두 실수를 입려기 32.765 3.987

  연산결과: 8.22

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

#include <stdio.h>

int main(void)
{
	double x = 0, y = 0, result = 0;

	printf("두 실수를 입력: ");
	scanf("%lf %lf", &x, &y);

	if (x > y)
	{
		result = x / y;
	}
	else if (x == y)
	{
		result = x * y;
	}
```

```
        else
        {
                result = x + y;
        }

        printf("연산 결과: %.2f₩n", result);

        return 0;
}
```

LAB

표준입력으로 받은 세 정수의 최대값을 출력

- 먼저 조건식 x> y 이 참이면 x 와 y 의 최대값, 거짓이면 x 와 y 의 최소값

  세 정수를 입력: 10 20 30

  최대 수: 30

```
// file: maxof3.c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int main(void)
{
        int x, y, z;

        printf("세 정수를 입력: ");
        scanf("%d %d %d", &x, &y, &z);

        if (x > y)
        {
                if (x > z)
                        printf("최대 수: %d₩n", x);
                else
                        printf("최대 수: %d₩n", z);
        }
        else
        {
                if (y > z)
                        printf("최대 수: %d₩n", y);
                else
                        printf("최대 수: %d₩n", z);
        }

        return 0;
}
```

6.3 다양한 선택 switch 문

1) switch 문장 개요

문장 if else 가 여러 번 반복되는 구문을 좀 더 간략하게 구현할 때 switch 문은 주어진 연산식이 문자형 또는 정수형이라면 그 값에 따라 case 의 상수값과 일치하는 부분의 문자들을 수행하는 선택 구문이다.

switch (exp) {....} 문은 표현식 exp 결과값 중에서 case 의 값과 일치하는 항목의 문장 stmt1 을 실행한 후 break 를 만나 종료한다. 연산식 exp 의 결과값은 반드시 문자 또는 정수여야한다. case 다음의 value 값은 변수가 올 수 없으며 상수식으로 그 결과가 정수 또는 문자 상수여야 하고 중복될 수 없다. default 는 선택적이므로 사용하지 않을 수 있다.

- 실행 순서

1. 표현식 exp 를 평가하여 그 값과 일치하는 상수값을 갖는 case 값을 찾아 case 내부의 문장을 실행한다.
2. break 를 만나면 switch 문을 종료한다. case 문의 내부문장을 실행하고 break 문이 없으면 break 문을 만나기 전까지 다음 case 의 내부로 무조건 이동하여 내부 문장을 실행한다.
3. 일치된 case 값을 만나지 못하여 default 를 만나면 default 내부의 문장을 실행한다. default 의 이후에 다른 case 가 없으면 break 는 생략 가능하다.
4. default 의 위치는 모든 case 뒤에 오는 것이 일반적이나, 어디에도 위치할 수 있으며 중간에 위치하면서 break 문이 없으면 하부 case 내부 문장을 무조건 실행한다.

switch 문에서 주의할 것 중 하나는 case 이후 정수 상수를 콤마로 구분하여 여러 개 나열할 수 없다는 것이다. case 문 내부에 break 문이 없다면 일치하는 case 문을 실행하고, break 문을 만나기 전까지 다음 case 내부 문장을 실행한다.

LAB

표준입력으로 받은 세 정수의 최대값을 출력

- 먼저 조건식 x > y 의 결과를 switch 문을 이용
- 조건연산자를 이용하여 두 수 중에서 최대값을 출력

  세 정수를 입력: 5 10 8

  최대값: 10

```
// file: simplemaxof3.c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int main(void)
{
        int x, y, z;

        printf("세 정수를 입력: ");
        scanf("%d %d %d", &x, &y, &z);

        switch ((x > y))
        {
        case 0:
                printf("최대 값: %d₩n", y > z ? y : z);
                break;

        case 1:
                printf("최대 값: %d₩n", x > z ? x : z);
                break;
        }

        return 0;
}
```

2) switch 연산식의 활용과 default 의 위치

일반적으로 switch 문에서 default 는 생략될 수 있으며, 그 위치도 제한이 없다. 다만 default 를 위치시킨 이후에 다른 case 가 있다면 break 를 반드시 입력한다.

- switch 문에서의 주의점

  - switch 연산식 결과는 정수형 또는 문자형이어야 한다.

  - 각 case 뒤에 나오는 식은 상수식이어야하며, 그 결과는 모두 달라야한다.

  - default 는 선택적으로 없거나 하나이며 다른 case 뒤에 있다면 break 가 필요하다.

  - 상수식에는 변수와 const 상수를 절대 사용할 수 없다.

LAB

표준입력으로 받은 세 정수의 최대값을 출력

- 삼원색을 표현하는 열거상수로 RED, GREEN, BLUE 를 정의
- 세 정수 (R[0],G[1],B[2]) 중의 하나를 입력
- switch 의 case 상수로 열거 상수를 이용

  세 정수 (R[0],G[1],B[2]) 중의 하나를 입력: 0

  Red

```
// enumswitch.c
#define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의

#include <stdio.h>

int main(void)
{
        enum color { RED, GREEN, BLUE };
        int input;

        printf("세 정수(R[0], G[1], B[2]) 중의 하나를 입력: ");
        scanf("%d", &input);

        switch (input) {
        case RED:
                printf("Red₩n");
                break;

        case GREEN:
                printf("Green₩n");
                break;
```

```
        case BLUE:
                printf("Blue₩n");
                break;


        default:
                printf("잘못된 입력₩n");
        }

        return 0;
}
```

CHAPTER 07 반복

7.1 반복 개요와 while 문

1) 반복 개요

반복은 같거나 비슷한 일을 여러 번 수행하는 작업이다. C 언어는 while, do while, for 세 가지 종류의 반복 구문을 제공한다.

- 반복문 while

  - 반복할 때 마다 조건을 따지는 반복문으로, 조건식이 반복몸체 앞에 위치한다.

```
while ( <반복조건> )
{
        //반복몸체 (loop body);
        <해야할 일>;
}
```

- 반복문 do while

  - 무조건 한 번 실행 한 후 조건을 검사하고 이때 조건식이 참(0 이 아니면)이면 반복을 더 실행한다.

```
do
{
        //반복몸체 (loop body);
        <해야할 일>;
} while ( <반복조건> );
```

- 반복문 for

  - 숫자로 반복하는 횟수를 제어하는 반복문으로 명시적으로 반복 횟수를 결정한다.

  for (<초기화>; <반복조건>; <증감>)

  {

  //반복몸체 (loop body);

  <해야할 일>;

  }

다음은 여러 섭씨온도를 화씨온도로 출력하는 반복문 소스이다.

```
// file: cel2far3.c
#include <stdio.h>

int main(void)
{
        double celcius = 12.46;

        printf("  섭씨(C)  화씨(F)₩n");
        printf("--------------------₩n");
        printf("%8.2lf %8.2lf₩n", celcius, 9.0 / 5 * celcius + 32);
        celcius += 10;
        printf("%8.2lf %8.2lf₩n", celcius, 9.0 / 5 * celcius + 32);
        celcius += 10;
        printf("%8.2lf %8.2lf₩n", celcius, 9.0 / 5 * celcius + 32);
        celcius += 10;

        return 0;
}
```

[실행결과]

```
섭씨 ( C )        화씨 ( F )

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

12.46             54.43

22.46             72.43

32.46             98.43
```

2) while 문장

문장 while (cond) stmt;는 반복조건인 cond 를 평가하여 0 이 아니면(참) 반복몸체인 stmt 를 실행하고 다시 반복조건 cond 를 평가하여 while 문 종료 시까지 반복한다.

예시)
```
int count = 1;
while (count <=3)
{
    printf("C 언어 재미있네요₩n");
    count++;
};
```

- 이 반복은 cond 가 0(거짓)이 될때까지 계속된다.
- 반복이 실행되는 stmt 를 반복몸체라 부르며, 필요하면 블록으로 구성될 수 있다.
- while 문은 for 나 do while 반복문보다 간단하며 모든 반복 기능을 수행할 수 있다.

반복 구문에서 반복 횟수를 제어하는 변수를 제어변수라 한다. 조건식 (count <=3)에서 상수 3 은 최대 반복횟수를 저장하는 상수이며, 반복몸체에서 제어변수 count 횟수만큼 반복을 위해 count 를 1 증가시키는 count++ 문장이 반드시 필요하다.

LAB

0 부터 20 까지의 3 의 배수 출력

- 정수를 모두 한 줄에 출력

  0 3 6 9 12 15 18

```c
// file: whilelab.c

#include <stdio.h>
#define MAX 20

int main(void)
{
        int n = 0;

        while (n <= MAX) {
                printf("%4d", n);
                n += 3;
        }
        puts("");

        return 0;
}
```

7.2 do while 문과 for 문

1) do while 문

while 문은 반복 전에 반복조건을 평가한다. 이와 달리 do while 문은 반복몸체 수행 후에 반복조건을 검사한다. 문장 do stmt; while (cond)는 가장 먼저 stmt 를 실행한 이후 반복조건인 cond 를 평가하여 0 이 아니면(참) 다시 반복몸체인 stmt;를 실행하고, 0 이면(거짓) do while 문을 종료한다.

```
do {
    stmt;
 } while (cond)
next;
```

```
do {
    printf("양의 정수 또는 0(종료)을 입력: ");
    scanf("%d",&input);
    ...
 } while (input != 0)
```

특히 반복 횟수가 정해지지 않고 입력 받은 자료값에 따라 반복 수행의 여부를 결정하는 구문에 유용하다. 반복몸체에 특별히 분기 구문이 없는 경우, do while 의 몸체는 적어도 한 번은 실행되는 특징이 있다. 이러한 반복문은 센티널 값 검사에 유용하게 사용된다.

다음은 1 에서부터 5 까지 1 씩 증가되는 값을 출력하는 프로그램을 do while 문을 이용해 작성한 프로그램이다.

```c
// file: dowhilenumber.c

#include <stdio.h>
#define MAX 5

int main(void)
{
        int n = 1;

        do
        {
                printf("%d₩n", n++);
        } while (n <= MAX);

        printf("₩n제어변수 n => %d₩n", n);

        return 0;
}
```

[실행결과]

1

2

3

4

5

제어변수 n >= 6

LAB

반복문 do while 문을 사용하여 백단위의 양의 정수를 입력 받아 각각 100 단위, 10 단위, 1 단위 값을 출력하는 프로그램을 작성한다.

- 정수는 100 에서 999 사이의 정수를 입력하며, 나누기 연산자 / 와 나머지 연산자 % 를 잘 활용하여 다음과 같이 출력
- 정수의 나누기 연산자 / 의 결과는 정수 몫으로 673 / 100 은 6
- 정수의 나머지 연산자 %의 결과는 나머지 값으로 673 % 100 은 73

  양의 정수[100~999] 입력: 853

  100 단위 출력: 8

  10 단위 출력: 5

  1 단위 출력: 3

```c
// file: forlab.c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int main(void)
{
	int input = 0, result = 0, digit = 0;
	int devider = 100;

	printf("양의 정수[100~999] 입력 : ");
	scanf("%d", &input);
	result = input;
	do
	{
		digit = result / devider;
		result %= devider;
		printf("%3d단위 출력: %d₩n", devider, digit);
		devider /= 10;
	} while (devider >= 1);

	return 0;
}
```

2) for 문

while 과 do while 구문은 단순하게 조건식에 따라 반복을 구현하지만 for 반복문은 반복에 대한 제어변수의 초기화와 증감을 일정한 영역에서 코딩하도록 지원한다.

반복문 for (init; cond; inc) stmt; 에서 init 에서는 주로 초기화가 이루어지며, cond 에서는 반복조건을 검사하고, inc 에서는 주로 반복을 결정하는 제어변수의 증감을 수행한다.

예시) for (i=1; i<=10; i++)

{

printf("%3d",i);

}

변수 i 와 같이 반복의 횟수를 제어하는 변수를 제어변수라 한다.

for(; ;)의 괄호 내부에서 세미콜론으로 구분되는 항목은 모두 생략될 수 있다. 그러나 두 개의 세미콜론은 반드시 필요하다. 반복조건 cond 를 제거하면 반복은 무한히 계속된다.

- for 문의 실행순서
  - 초기화를 위한 init 를 실행한다. 이 init 는 단 한번만 수행된다.
  - 반복조건 검사 cond 를 평가해 0 이아닌 결과값(참)이면 반복문의 몸체에 해당하는 문장 stmt 를 실행한다. 그러나 조건검사 cond 가 결과값이 0(거짓)이면 for 문을 종료하고 다음 문장 next 를 실행한다.
  - 반복몸체인 stmt 를 실행한 후 증감연산 inc 를 실행한다.
  - 다시 반복조건인 cond 를 검사하여 반복한다.

다음 for 문은 오른쪽의 while 문과 같은 기능을 수행한다.

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

```
int i = 0;
while ( i <=10 ) {
        printf("%3d",i);
        i++;
}
```

10 도씩 증가하는 3 개의 섭씨온도(celcius)를 화씨온도로 변환하여 출력하는 소스를 작성해보고자 한다. 다음은 섭씨와 화씨온도를 출력할 반복횟수는 매크로 상수 MAX 로 정의하고, 섭씨 온도의 증가 값은 매크로 상수 INCREMENT 로 정의하여, 섭씨온도 celcius 를 12.46 을 시작으로 3 개의 화씨온도를 각각 출력하는 for 문이다.

```
// file: forcel2far3.c

#include <stdio.h>
#define MAX 3
#define INCREMENT 10

int main(void)
{
        double celcius = 12.46;

        printf("  섭씨(C)  화씨(F)₩n");
        printf("--------------------₩n");

        for (int i = 1; i <= MAX; i++, celcius += INCREMENT)
        {
                printf("%8.2lf %8.2lf₩n", celcius, 9.0 / 5 * celcius + 32);
        }

        return 0;
}
```

[실행결과]

```
섭씨 ( C )        화씨 ( F )
------------------------------
12.46             54.43
22.46             72.43
32.46             98.43
```

3) for 문의 활용

for 문을 이용하여 1 에서 10 까지의 합을 구하는 프로그램을 작성하고자 한다. 먼저 제어변수 i 값을 계속 합하여 변수 sum 에 누적시키고 제어변수 i 와 합을 저장하는 변수 sum 의 초기값 지정이 필요하다. 즉 i 를 1 로 설정하고, sum 을 0 으로 초기화하는 문장을

for (i=1, sum=0; i<10; i++)와 같이 초기화 부분에 콤마연산자를 이용하여 나열할 수 있다.

반복몸체의 문장 sum = sum + i ; 는 축약대입연산자 +=를 이용하여 sum+=i 로 축약 가능하다. 만일 sum+=i++로 증가연산자를 이용한다면 for 문의 증감부분은 생략될 수 있다. 그러나 sum+=++i 로 증가연산자를 이용한다면 제어변수 i 를 0 에서 9 까지 반복해야 한다.

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

int main(void)
{
	int i, sum;

	for (i = 1, sum = 0; i <= 10; i++) //++i도 가능
		sum += i; // sum = sum + i;
	printf("1에서 10까지 합: %3d₩n", sum);

	for (i = 1, sum = 0; i <= 10;)
		sum += i++;
	printf("1에서 10까지 합: %3d₩n", sum);

	for (i = 0, sum = 0; i <= 9;)
		sum += ++i;
	printf("1에서 10까지 합: %3d₩n", sum);

	for (i = 1, sum = 0; i <= 10; sum += i++);//반복몸체가 없는 for 문
	printf("1에서 10까지 합: %3d₩n", sum);

	return 0;
}
```

[실행결과]

1 에서 10 까지 합: 55

1 에서 10 까지 합: 55

1 에서 10 까지 합: 55

1 에서 10 까지 합: 55

4) for 문과 while 문의 비교

for 문은 주로 반복횟수를 제어하는 제어변수를 사용하며 초기화와 증감부분이 있는 반복문에 적합하다. 반면 while 문은 반복횟수가 정해지지 않고 특정한 조건에 따라 반복을 결정하는 구문에 적합하다.

다음은 1 에서부터 표준입력으로 받은 양의 정수까지의 합을 for 과 while 로 각각 구한 프로그램이다.

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

#include <stdio.h>

int main(void)
{
	int i, sum, max;

	printf("1에서부터 정수까지의 합을 구할 양의 정수 하나 입력: ");
	scanf("%d", &max);

	for (i = 1, sum = 0; i <= max; i++) //++i도 가능
		sum += i; // sum = sum + i;
	printf("₩nfor 문으로 구한 1에서 %d까지 합: %3d₩n", max, sum);

	i = 1, sum = 0;
	while (i <= max)
	{
		sum += i;	// sum = sum + i;
		i++;		// ++i도 가능
	}
	printf("while 문으로 구한 1에서 %d까지 합: %3d₩n", max, sum);

	return 0;
}
```

[실행결과]

1 에서부터 정수까지의 합을 구할 양의 정수 하나 입력: 20

for 문으로 구한 1 에서 20 까지의 합: 210

while 문으로 구한 1 에서 20 까지의 합: 210

LAB

반복문 for 을 사용하여 2 단부터 9 단까지의 구구단의 제목을 출력하는 프로그램

결과: ===구구단 출력===

2 단 출력

3 단 출력

4 단 출력

5 단 출력

6 단 출력

7 단 출력

8 단 출력

9 단 출력

```c
// file: forlab.c

#include <stdio.h>
#define MAX 9

int main(void)
{
        printf("=== 구구단 출력 ===₩n");
        for (int i = 2; i <= MAX; i++)
        {
                printf("%6d단 출력₩n", i);
        }

        return 0;
}
```

7.3 분기문

1) 분기문 개요

분기문은 정해진 부분으로 바로 실행을 이동(jump)하는 기능을 수행한다. C 가 지원하는 분기문으로는 break, continue, goto, return 문이 있다.

2) 반복의 중단 break

반복내부에서 반복을 종료하려면 break 문장을 사용한다. 중첩된 반복에서의 break 는 자신이 속한 가장 근접한 반복에서 반복을 종료한다.

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

#include <stdio.h>

int main(void)
{
	int input;

	while (1)
	{
		printf("정수[음수, 0(종료), 양수]를 입력 후 [Enter] : ");
		scanf("%d", &input);
		printf("입력한 정수 %d₩n", input);
		if (input == 0)
			break;
	}
	puts("종료합니다.");

	return 0;
}
```

```
[실행결과]

정수[음수, 0(종료), 양수]를 입력 후 [Enter] : 10

입력한 정수: 10

정수[음수, 0(종료), 양수]를 입력 후 [Enter] : -5

입력한 정수: -5

정수[음수, 0(종료), 양수]를 입력 후 [Enter] : 0

입력한 정수: 0

종료합니다.
```

2) 반복의 계속 continue

continue 문은 반복의 시작으로 이동하여 다음 반복을 실행하는 문장이다. 즉 continue 문은 continue 문이 위치한 이후의 반복몸체의 나머지 부분을 실행하지 않고 다음 반복을 계속 유지하는 문장이다. 반복문 while 과 do while 반복 내부에서 w 를 만나면 조건검사로 이동하여 실행한다. 반복문 for 문에서 continue 문을 만나면 증감 부분으로 이동하여 다음 반복 실행을 계속한다.

다음은 continue 문을 이용하여 1 에서 15 까지 정수 중에서 3 으로 나누어 떨어지지 않는 수를 출력하는 프로그램이다.

```c
// file: continue.c

#include <stdio.h>

int main(void)
{
	const int MAX = 15;

	printf("1에서 %d까지 정수 중에서 3으로 나누어 떨어지지 않는 수₩n", MAX);
	for (int i = 1; i <= MAX; i++)
	{
		if (i % 3 == 0) // (!(i % 3))
			continue;
		printf("%3d", i);
	}
	puts("");

	return 0;
}
```

[실행결과]

1 에서 15 까지 정수 중에서 3 으로 나누어 떨어지지 않는 수

1 2 4 5 7 8 10 11 13 14

3) goto 와 무한반복

goto 문은 레이블이 위치한 다음 문장으로 실행순서를 이동하는 문장이다. 레이블은 식별자와 콜론을 이용하여 지정한다. goto 문을 적절히 이용하면 반복문처럼 이용할 수 있으나 goto 문은 프로그램의 흐름을 어렵고 복잡하게 만들 수 있으므로 사용하지 않는 것이 바람직하다.

반복문에서 무한히 반복이 계속되는 것을 무한반복이라 한다. while 과 do while 은 반복조건이 아예 없으면 오류가 발생하다. for 문에서 for(init; ; inc)와 같이 반복조건에 아무것도 없으면 오류없이 무한반복이 실행된다.

LAB

분기문 continue 문을 사용하여 1 부터 15 까지의 정수 중에서 5 의 배수가 아닌 수를 출력하는 프로그램

- 정수는 모두 한 줄에 계속 출력

  1 에서 15 까지 정수 중에서 5 로 나누어 떨어지지 않는 수

  1 2 3 4 6 7 8 9 11 12 13 14

```c
// file: continuelab.c

#include <stdio.h>

int main(void)
{
        const int MAX = 15;

        printf("1에서 %d까지 정수 중에서 5로 나누어 떨어지지 않는 수₩n", MAX);
        for (int i = 1; i <= MAX; i++)
        {
                if (!(i % 5))
                        continue;
                printf("%3d", i);
        }
        puts("");

        return 0;
}
```

LAB

반복문 for 문을 사용하여 2 단부터 9 단까지의 구구단을 출력하는 프로그램

```c
// file: mtable.c

#include <stdio.h>
#define MAX 9

int main(void)
{
        printf("=== 구구단 출력 ===₩n");
        for (int i = 2; i <= MAX; i++)
        {
                printf("%6d단 출력₩n", i);
                for (int j = 2; j <= MAX; j++)
                        printf("%d*%d = %2d ", i, j, i*j);
                printf("₩n");
        }

        return 0;
}
```

CHAPTER 08 포인터 기초

8.1 포인터 변수와 선언

1) 메모리 주소와 주소연산자 &

메모리 고안은 8 비트인 1 바이트마다 고유한 주소(address)가 있다. 메모리 주소는 0 부터 바이트마다 1 씩 증가한다. 메모리 주소는 저장 장소인 변수이름과 함께 기억 장소를 참조하는 또 다른 방법이다.

주소는 변수이름과 같이 저장장소를 참조하는 하나의 방법이다. 지금까지 함수 scanf()를 사용하면서 자료의 값을 저장하기 위해 인자를 '&변수이름'으로 사용하였다. 바로 &가 피연산자인 변수의 메모리 주소를 반환하는 주소연산자이다.

- 즉 함수 scanf()에서 입력값을 저장하는 변수의 주소값이 인자의 자료형이다.
- 그러므로 함수 scanf()에서 일반 변수 앞에는 주소연산자 &를 사용해야 한다.

변수의 주소값은 형식제어문자 %u 또는 %d 로 직접 출력할 수 있다. 그러나 경고를 방지하기 위해 주소값을 int 혹은 unsigned 로 변환하여 출력한다.

- &연산자는 '&변수'와 같이 피연산자 앞에 위치하는 전위연산자로 변수에만 사용할 수 있다.
- '&32'와 '&(3+4)'와 같이 상수나 표현식에는 사용할 수 없다.

2) 포인터 변수 개념과 선언

변수의 주소는 포인터 변수에 저장할 수 있다. 그러나 일반 변수에 저장하면 주소값이라는 의미가 없어지므로 변수의 주소값은 반드시 포인터 변수에 저장해야 한다. 즉 포인터 변수는 주소값을 저장하는 변수로 일반 변수와 구별되며 선언방법이 다르다.

포인터 변수는 일반 변수와 선언 방법이 다른데, 포인터 변수 선언에서 자료형과 포인터 변수 이름 사이에 연산자 *(asterisk)를 삽입한다. 즉 다음 변수선언에서 ptrint, ptrshort, ptrchar 은 모두 포인터 변수이며 간단히 포인터라고도 부른다.

```
예시) 자료형     *변수이름;

      int       *ptrint;

      int       *ptrshort;

      char      *ptrchar;
```

위 포인터 변수선언에서도 보듯이 변수 자료형이 다르면 그 변수의 주소를 저장하는 포인터의 자료형도 달라야 한다. 즉 어느 변수의 주소값을 저장하려면 반드시 그 변수의 자료유형과 동일한 포인터 변수에 저장해야 한다. 포인터 변수선언에서 포인터를 의미하는 *는 자료형과 변수이름 사이에만 위치하면 된다.

- 포인터 변수와 일반 변수의 주소 저장

```
int data = 100;          // 변수 data 는 정수 int 를 저장하는 일반 변수

int *ptrint ;            // 변수 ptrint 는 정수 int 를 저장하는 일반변수의 주소를
                         저장하는 포인터 변수

ptrint = &data;          // 포인터 ptrint 는 이제 data 주소값을 갖는다
```

포인터 변수도 선언된 후 초기값이 없으면 의미 없는 쓰레기 값이 저장된다. 포인터 변수에 data 주소를 대입하려면 주소연산자 &를 사용한 연산식 &data 를 이용한다. 그러므로 문장 ptrint = &data; 는 포인터 변수 ptrint 에 변수 data 의 주소를 저장하는 문장이다. 포인터 변수는 가리키는 변수의 종류에 상관없이 크기가 모두 4 바이트이다.

다음은 포인터 변수선언과 주소값을 대입하는 프로그램이다.

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

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

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

        return 0;
}
```

[실행결과]

```
변수명          주소값          저장값

==============================

  data          0024FB44        100

  ptrint        0024FB38        0024FB44
```

LAB

자료형 char, int, double 의 변수를 각각 선언하여 적당한 값을 저장하고, 다시 그 변수의 주소를 저장하기 위해 자료형 char, int, double 의 포인터 변수를 각각 선언해 그 주소값과 저장한 후, 주소값과 저장값을 출력하는 프로그램을 작성한다.

- char 포인터 변수 선언: char *pc
- int 포인터 변수 선언: int *pm
- double 포인터 변수 선언: double *px

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

int main(void)
{
        char c = '@';
        char *pc = &c;
        int m = 100;
        int *pm = &m;
        double x = 5.83;
        double *px = &x;

        printf("변수명    주소값     저장값₩n");
        printf("-------------------------₩n");
        printf("%3s %12p %9c₩n", "c", pc, c);
        printf("%3s %12p %9d₩n", "m", pm, m);
        printf("%3s %12p %9f₩n", "x", px, x);

        return 0;
}
```

[실행결과]

```
변수명          주소값          저장값

==============================

  c             002DFE0B        @

  m             002DFDF0        100

  x             002DFDD4        5.830000
```

8.2 간접 연산자 *와 포인터 연산

1) 다양한 포인터 변수 선언과 간접 연산자*

여러 개의 포인터 변수를 한 번에 선언하기 위해서는 다음과 같이 콤마 이후에 변수마다 *를 앞에 기술해야 한다.

예시) int *ptr1, *ptr2, *ptr3 ; // ptr1, pt2, ptr3 모두 int 형 포인터임

int *ptr1, ptr2, ptr3 // ptr1 은 int 형 포인터이나 ptr2 와 ptr3 은 변수임

포인터 변수는 다른 일반변수와 같이 지역변수로 선언하는 경우, 초기값을 대입하지 않으면 쓰레기값이 들어가므로 포인터 변수에 지정할 특별한 초기값이 없는 경우에 0 번 주소값인 NULL 로 초기값을 지정한다.

예시) int *ptr = NULL;

이 NULL 은 헤더파일 <stdio.h>에 다음과 같이 정의되어 있는 포인터 상수로서 0 번지의 주소값을 의미한다. 그러므로 ptr 을 출력하면 실제로 0 이 출력된다. 여기서 (void *)는 아직 결정되지 않은 자료형의 주소를 나타낸다. 또한 자료유형 (void *)는 아직 유보된 포인터이므로 모든 유형의 포인터 값을 저장할 수 있는 포인터 형이다.

예시) #define NULL ((void *) 0)

포인터를 사용하는 이유를 알아보면, 포인터는 변수를 참조할 수 있는 또 다른 방법을 제공한다는 것이다. 즉 포인터 변수가 갖는 주소로 그 주소의 원래 변수를 참조할 수 있다. 포인터 변수가 가리키고 있는 변수를 참조하려면 간접연산자 * 를 사용한다.

- 간접연산자 * 와 간접참조

```
int data1 = 100 , data2;
int *p ;
printf("간접참조 출력: %d ₩n", *ptrint);
*ptrint = 200 ;
```

간접 연산자를 이용한 *ptrint 는 포인터 ptrint 가 가리키고 있는 변수 자체를 의미한다. 즉 포인터 ptrint 가 가리키는 변수가 data 라면 *ptrint 는 변수 data 를 의미한다.

변수 data 자체를 사용해 자신을 참조하는 방식을 직접참조라 한다면, *ptrint 를 이용해서 변수 data 를 참조하는 방식을 간접참조라 한다.

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

int main(void)
{
        int data = 100;
        char ch = 'A';
        int *ptrint = &data;
        char *ptrchar = &ch;
        printf("간접참조 출력: %d %c₩n", *ptrint, *ptrchar);

        *ptrint = 200; //변수 data를 *ptrint로 간접참조하여 그 내용을 수정
        *ptrchar = 'B'; //변수 ch를 *ptrchar로 간접참조하여 그 내용을 수정
        printf("직접참조 출력: %d %c₩n", data, ch);

        return 0;
}
```

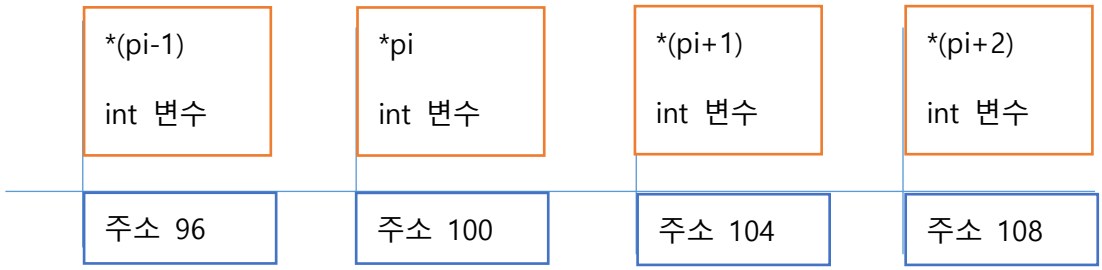
[실행결과]

간접참조 출력: 100 'A'

직접참조 출력: 200 'B'

2) 포인터 변수의 연산

포인터 변수는 간단한 더하기와 뺄셈 연산으로 이웃한 변수의 주소 연산을 수행할 수 있다. 포인터에 저장된 주소값의 연산으로 이웃한 이전 또는 이후의 다른 변수를 참조할 수 있다.



위 그림과 같이 int 형 포인터 pi 에 저장된 주소값이 100 이라고 가정한다면 (pi+1)은 101 이 아니라 주소값 104 이다. 즉 (pi+1)은 pi 가 가리키는 다음 int 형의 주소를 의미한다. 그러므로 (pi+1)은 int 형의 바이트 크기인 4 만큼 증가한 주소값 104 가 되는 것이다. 이처럼 더하기와 빼기 연산에는 포인터 변수가 피연산자로 참여할 수 있다. 그러나 나누기와 곱하기는 오류를 발생시킨다.

다음은 포인터 변수의 간단한 덧셈, 뺄셈 연산이다.

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

int main(void)
{
        char *pc = (char *)100;  //가능하나 잘 이용하지 않음
        int *pi = (int *)100;    //가능하나 잘 이용하지 않음
        double *pd = (double *)100;//가능하나 잘 이용하지 않음
        pd = 100;                //경고발생

        printf("%u  %u  %u₩n", (int)(pc - 1), (int)pc, (int)(pc + 1));
        printf("%u  %u  %u₩n", (int)(pi - 1), (int)pi, (int)(pi + 1));
        printf("%u  %u  %u₩n", (int)(pd - 1), (int)pd, (int)(pd + 1));

        return 0;
}
```

[실행결과]

```
99      100     101

96      100     104

92      100     108
```

LAB

정수 int 자료형 두 변수 m, n 에 저장된 두 값을 서로 교환하는 프로그램을 작성해보자. 제한 사항은 임시변수인 dummy 를 사용하고, 포인터 변수 p 를 사용하거나 변수 m, n 자체는 사용하지 않으며, 주소값 &m 과 &n 만을 사용하도록 한다.

- 포인터 변수 선언 int *p = &m;으로 *p 는 m 자체를 의미함
- 마찬가지로 대입문장 p=&m; 으로 *p 는 n 자체를 의미함

[실행결과]

100 200

200 100

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

int main(void)
{
        int m = 100, n = 200, dummy;
        printf("%d %d₩n", m, n);

        //변수 m과 n을 사용하지 않고 두 변수를 서로 교환
        int *p = &m; //포인터 p가 m을 가리키도록
        dummy = *p;  //변수 dummy에 m을 저장
        *p = n;           //변수 m에 n을 저장
        p = &n;      //포인터 p가 n을 가리키도록
        *p = dummy;  //변수 n에 dummy 값 저장

        printf("%d %d₩n", m, n);

        return 0;
}
```

8.3 포인터 형변환과 다중 포인터

1) 내부 저장 표현과 포인터 변수의 형변환

변수 value 에 16 진수 0x61626364 를 저장하고자 할 때, 주소는 int value = 0x61626364 ; int *pi = &value; 이다.

포인터 변수는 동일한 자료형끼리만 대입이 가능하다. 만일 대입문에서 포인터의 자료형이 다르면 경고가 발생한다.

- 포인터의 형변환 경고

  int value = 0x61626364 ;

  int *pi = &value ;

  char *pc = &value ; //경고

*pc 로 수행하는 간접참조는 pc 가 가리키는 주소에서부터 1 바이트 크기의 char 형 자료를 참조한다는 것을 의미한다.

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

int main(void)
{
        int value = 0x61626364;
        int *pi = &value;
        char *pc = (char *)&value; //char *pc = &value;

        printf("변수명   저장값      주소값₩n");
        printf("---------------------------₩n");
        printf(" value   %0#x  %p₩n", value, pi); //정수 출력

        //문자 포인터로 문자 출력 모듈
        for (int i = 0; i <= 3; i++)
        {
                char ch = *(pc + i);
                printf("*(pc+%d)  %0#6x  %2c  %p₩n", i, ch, ch, pc + i);
        }

        return 0;
}
```

2) 다중 포인터와 증감 연산자의 활용

포인터 변수의 주소값을 갖는 변수를 이중 포인터라 한다. 다시 이중 포인터의 주소값을 갖는 변수를 삼중 포인터라 할 수 있다. 이러한 포인터의 포인터를 다중 포인터라고 하며 변수 선언에서 *를 여러 번 이용하여 다중 포인터 변수를 선언한다.

- 다음 소스에서 pi 는 포인터이며, 포인터 변수 pi 의 주소값을 저장하는 변수 dpi 는 이중 포인터이다.

  int i = 20 ;

  int *pi = &i ;

  int **dpi = &pi ;

다중 포인터 변수를 이용하여 일반 변수를 참조하려면 가리킨 횟수 만큼 간접연산자를 이용한다. 즉 이중 포인터 변수 dpi 는 **dpi 가 바로 변수 i 이다.

- 다중 포인터의 이용

  *pi = i+2 ;

  **dpi = *pi + 2;

3) 간접 연산자와 증감 연산자 활용

간접 연산자 *는 전위 연산자로 연산자 우선순위가 2 위이며, 증감연산자 ++, --는 전위이면 2 위이고, 후위이면 1 위이다.

우선순위	단항 연산자	설명	결합성(계산방향)
1	a++, a--	후위 증가, 후위 감소	-> (좌에서 우로)
2	++a, --a, &, *	전위 증가, 전위 감소 주소 간접 또는 역참조	<- (우에서 좌로)

- *p++는 *(p++)으로 (*p)++와 다르다.
- ++*p 와 ++(*p)는 같다.
- *++p 와 *(++p)는 같다.

연산자		결과값	연산 후 *p 의 값	연산 후 p 증가
*p++	*(p++)	*p: p 의 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: p 의 간접참조 값	*p 가 1 증가	p: 없음
++*p	++(*p)	*p+1: p 의 간접참조 값에 1 증가	*p 가 1 증가	p: 없음

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

int main(void)
{
        int i;
        int *pi = &i;            //포인터 선언
        int **dpi = &pi;    //이중포인터 선언

        *pi = 5;
        *pi += 1; //*pi = *pi + 1와 같음
        printf("%d₩n", i);

        // 후위 연산자 pi++는 전위 연산자보다 *pi보다 빠름
        printf("%d₩n", (*pi)++); //*pi++ 는 *(pi++)으로 (*pi)++과 다름
        printf("%d₩n", *pi);

        *pi = 10;
        printf("%d₩n", ++*pi); //++*pi과 ++(*pi)는 같음
        printf("%d₩n", ++**dpi); //++**dpi과 ++(**dpi)는 같음
        printf("%d₩n", i);

        return 0;
}
```

[실행결과]

```
6
6
7
11
12
12
```

4) 포인터 상수

키워드 const 를 이용하는 변수 선언은 변수를 상수로 만들듯이 포인터 변수도 포인터 상수로 만들 수 있다.

- 키워드 const 가 가장 먼저 나오는 선언은 *pi 를 사용해 포인터 pi 가 가리키는 변수인 i 를 수정할 수 없도록 하는 상수 선언 방법이다. 즉 간접 연산식 *pi 를 상수로 만들면 *pi 를 1-value 로 사용할 수 없다
- 키워드 const 가 중간에 나오는 선언은 1 번과 동일한 문장으로 간접 연산식 *pi 를 상수로 만드는 방법이다.
- 키워드 const 가 int *와 변수 pi 사이에 나오는 선언은 포인터 pi 에 저장되는 초기 주소값을 더 이상 수정할 수 없도록 하는 상수 선언 방법이다. 즉 이 문장은 포인터 변수 pi 자체를 상수로 만드는 방법으로 선언 이후 pi 를 1-value 로 사용할 수 없다.

```c
/* constptr.c */
#include <stdio.h>

int main()
{
        int i = 10, j = 20;
        const int *p = &i; //*p가 상수로 *p로 수정할 수 없음
        //*p = 20; //오류 발생
        p = &j;
        printf("%d₩n", *p);

        double d = 7.8, e = 2.7;
        double * const pd = &d;
        //pd = &e; //pd가 상수로 다른 주소 값을 저장할 수 없음
        *pd = 4.4;
        printf("%f₩n", *pd);

        return 0;
}
```

[실행결과]

20

4.400000

LAB

자료형 double 로 선언된 두 x 와 y 에 표준입력으로 두 실수를 입력 받아 두 실수의 덧셈 결과를 출력하는 프로그램을 작성해본다. 제한 사항은 두 변수 x 와 y 는 선언만 수행하며, 포인터 변수인 px 와 py 만을 사용하여 모든 과정을 코딩한다.

- double 포인터 변수 px 선언: double *px = &x ;
- double 포인터 변수 py 선언: double *py = &y ;

[실행결과]

두 실수 입력: 3.874 7.983

3.87 + 7.98 = 11.86

```c
// file: sumpointer.c
#define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의
#include <stdio.h>

int main(void)
{
	double x, y;
	double *px = &x;
	double *py = &y;

	//포인터 변수 px와 py를 사용
	printf("두 실수 입력: ");
	scanf("%lf %lf", px, py);
	//합 출력
	printf("%.2f + %.2f = %.2f₩n", *px, *py, *px + *py);

	return 0;
}
```

CHAPTER 09 배열

9.1 배열 선언과 초기화

1) 배열의 필요성과 정의

배열(array)은 여러 변수들이 같은 배열이름으로 일정한 크기의 연속된 메모리에 저장되는 구조이다. 배열을 이용하면 변수를 일일이 선언하는 번거로움을 해소할 수 있고, 배열을 구성하는 각각의 변수를 참조하는 방법도 간편하며, 반복 구문으로 쉽게 참조할 수 있다.

배열은 동일한 자료 유형이 여러 개 필요한 경우에 유용한 자료 구조이다. 즉 배열은 한 자료유형의 저장공간인 원소를 동일한 크기로 지정된 배열크기만큼 확보한 연속된 저장공간이다. 배열을 구성하는 각각의 항목을 배열의 원소라 한다. 배열에서 중요한 요소는 배열이름, 원소 자료유형, 배열 크기이다. 배열원소는 첨자 번호라는 숫자를 이용해 쉽게 접근할 수 있다.

2) 배열 선언과 원소 참조

배열선언은 int data[10] ; 과 같이 원소자료유형 배열이름[배열크기] ; 로 한다. 배열선언 시 초기값 지정이 없다면 반드시 배열크기는 양의 정수로 명시되어야 한다. 배열 크기를 지정하는 부분에는 양수 정수로 리터럴 상수와 매크로 상수 또는 이들의 연산식이 올 수 있다. 그러나 변수와 const 상수로는 배열의 크기를 지정할 수 없다.

배열선언 후 배열원소에 접근하려면 배열이름 뒤에 대괄호 사이 첨자를 이용한다. 배열에서 유효한 첨자의 범위는 0 부터 (배열크기-1)까지이며, 첨자의 유효 범위를 벗어나 원소를 참조하면 문법오류 없이 실행오류가 발생한다.

배열 선언 시 대괄호 안의 수는 배열 크기이다. 그러나 선언 이후 대괄호 안의 수는 원소를 참조하는 번호인 첨자이다.

반복 for 문의 제어변수를 0 에서 시작하여 배열크기보다 작을 때까지 출력을 반복한다.

- 배열원소 출력을 위한 반복 구문

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

3) 배열 초기화

C 언어는 배열을 선언하면서 동시에 원소값을 손쉽게 저장하는 배열선언 초기화 방법을 제공한다. 배열선언 초기화 구문은 배열선언을 하면서 대입연산자를 이용하며 중괄호 사이에 여러 원소값을 쉼표로 구분하여 기술하는 방법이다.

- 배열 초기화 구문

  원소자료형 배열이름[배열크기] = {원소값 1, 원소값 2, ... } ;

- 중괄호 사이에는 명시된 배열크기를 넘지 않게 원소값을 나열할 수 있다.
- 배열크기는 생략할 수 있으며, 생략하면 자동으로 중괄호 사이에 기술된 원소 수가 배열크기가 된다.
- 원소값을 나열하기 위해 콤마를 사용하고 전체를 중괄호로 묶는다.

만일 배열크기가 초기값 원소 수보다 크면 지정하지 않은 원소의 초기값은 자동으로 모두 기본값으로 저장된다. 즉 정수형은 0, 실수형은 0.0 그리고 문자형은 '₩0'인 널문자가 자동으로 채워진다.

배열크기가 초기값 원소 수보다 작으면 배열 저장공간을 벗어나므로 문법오류가 발생한다.

- 배열크기가 지정된 경우의 초기값 수

```
int dist[5] = {12, 25, 17, 55, 57, 71 } ;       //오류
int dist[5] = {0} ;            // 지정한 배열크기보다 초기값 수가 적으면 모두 0 으로
                                  채워지므로 모든 배열원소가 0 으로 채워진다.
```

LAB

자료형 int 로 선언된 배열 input 에서 표준입력으로 받은 정수를 순서대로 저장하여 출력하는 프로그램. 배열 input 을 선언하면서 초기화로 모두 0 을 저장하며, 표준입력으로 받은 정수는 0 이 입력될 때까지 저장하도록 한다.

- 입력된 배열은 while 문을 사용하여 마지막 0 값 이전까지 다음 결과와 같이 출력

  [실행결과]

  배열에 저장할 정수를 여러 개 입력하시오. 0 을 입력하면 입력을 종료합니다.

  30　　26 65 39 87 76 0

  30　　26 65 39 87 76

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

int main(void)
{
	//초기화로 모든 원소에 0을 저장
	int input[20] = { 0 };

	printf("배열에 저장할 정수를 여러 개 입력하시오.");
	printf(" 0을 입력하면 입력을 종료합니다.₩n");
	int i = 0;
	do {
		scanf("%d", &input[i]);
	} while (input[i++] != 0);

	i = 0;
	while (input[i] != 0) {
		printf("%d ", input[i++]);
	}
	puts("");

	return 0;
}
```

9.2 이차원과 삼차원 배열

1) 이차원 배열 선언과 사용

이차원 배열은 테이블 형태의 구조를 나타낼 수 있으므로 행과 열의 구조로 표현할 수 있다.

이차원 배열선언은 2 개의 대괄호가 필요하다. 첫 번째 대괄호에는 배열의 행 크기, 두번째는 배열의 열 크기를 지정한다. 배열선언 시 초기값을 저장하지 않으면 반드시 행과 열의 크기는 명시되어야 한다.

- 이차원 배열선언의 구문

  원소자료형 배열이름[배열행크기][배열열크기] ;

이차원 배열에서 각 원소를 참조하기 위해서는 2 개의 첨자가 필요하다. 배열선언 int td[2][3] ; 으로 선언된 배열 td 에서 첫번째 원소는 [0][0]으로 참조한다. 일차원 배열과 같이 이차원 배열원소를 참조하기 위한 행 첨자는 0 에서 (행크기-1)까지 유효하다. 마찬가지로 열 첨자는 0 에서 (열크기-1)까지 유효하다.

이차원 배열은 첫 번째 행 모든 원소가 메모리에 할당된 이후에 두 번째 행의 원소가 순차적으로 할당된다. c 언어와 같은 배열의 이러한 특징을 행 우선 배열이라 한다.

2) 이차원 배열 초기화

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

- 이차원 배열선언 초기화

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

이차원 배열선언 초기값 지정의 다른 방법으로는 일차원 배열과 같이 하나의 중괄호로 모든 초기값을 쉼표로 분리하는 방법이다. 이차원 배열선언 초기값 지정에서도 첫 번째 대괄호 내부의 행의 크기는 명시하지 않을 수 있다.

이차원 배열의 총 배열원소 수보다 적게 초기값이 주어지면 나머지는 모두 기본값인 0, 0.0 또는 ₩0'이 저장된다.

9.3 배열과 포인터 관계

1) 일차원 배열과 포인터

배열 score 에서 배열이름 score 자체가 배열 첫 원소의 주소값인 상수이다.

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

- 배열이름 score 는 배열 첫 번째 원소의 주소를 나타내는 상수로 &score[0]와 같다. 간접연산자를 이용한 *score 는 변수 score[0]와 같다.
- 배열이름 score 가 포인터 상수로 연산식 (score + 1)이 가능하다. 이것을 확장하면 (score+i)는 &score[i]이다.
- 마찬가지로 간접연산자를 이용한 *score 는 변수 score[0]인 것을 확장하면 *(score + i)는 score[i]와 같다.
- 간접연산자 *를 사용한 연산식 *(score+ i)는 배열 score 의 (i+1)번째 배열원소로 score[i]와 같다.

참조연산자 *의 우선순위는 ++p 의 전위 증감연산자와 같고, 괄호나 p++의 후위 증감연산자보다 낮다. 연산식 ++*p 는 ++(*p)로 포인터 p 가 가리키는 값을 1 증가시킨 후 참조한다.

연산자		결과값	연산 후 *p 의 값	연산 후 p 증가
++*p	++*(p)	*p + 1: p 의 간접참조 값에 1 증가	*p 가 1 증가	p: 없음
*p++	*(p++)	*p: p 의 간접참조 값	변동 없음	p+1: p 다음 주소
--*p	--(*p)	*p-1: p 의 간접참조 값에 1 감소	*p 가 1 감소	p: 없음
(*p)--		*p: p 의 간접참조 값	*p 가 1 감가	p+1: p 다음 주소

2) char 배열을 int 자료형으로 인식

포인터 변수는 동일한 자료형끼리만 대입이 가능하다. 만일 대입문에서의 포인터의 자료형이 다르면 경고가 발생한다. 포인터 변수는 자동으로 형변환이 불가능하며 필요하면 명시적으로 형변환을 수행할 수 있다.

*pi 로 수행하는 간접참조는 pi 가 가리키는 주소에서부터 4 바이트 크기의 int 형 자료를 참조한다는 것을 의미한다. 즉 동일한 메모리의 내용과 주소로부터 참조하는 값이 포인터의 자료형에 따라 달라진다.

3) 이차원 배열과 포인터

이차원 배열에서 배열이름인 td 는 포인터 상수 td[0]를 가리키는 포인터 상수이다.

```
int td[][3] = {{8,5,3}, {3,4,1}} ;
```

배열이름 td 는 이차원 배열을 대표하는 이중 포인터이며, sizeof(td)는 배열전체의 바이트 크기를 반환한다. td[0][0]의 값을 20 으로 수정하려면 **td=20; 문장을 이용할 수 있다.

```c
// fiel: tdaryptr.c
#include <stdio.h>

#define ROW 2
#define COL 3

int main(void)
{
	int td[][COL] = { { 8, 5, 4 }, { 2, 7, 6 } };

	**td = 10;		//td[0][0] = 10;
	*td[1] = 20;	//td[1][0] = 20;

	for (int i = 0, cnt = 0; i < ROW; i++)
	{
		for (int j = 0; j < COL; j++, cnt++)
		{
			printf("%d %d %d, ", *(*td + cnt), *(td[i] + j), *(*(td +
i) + j));
		}
		printf("₩n");
	}

	printf("%d, %d, %d₩n", sizeof(td), sizeof(td[0]), sizeof(td[1]));
	printf("%p, %p, %p₩n", td, td[0], td[1]);
	printf("%p, %p₩n", &td[0][0], &td[1][0]);

	return 0;
}
```

[실행결과]

10 10 10, 5 5 5, 4 4 4

20 20 20, 7 7 7, 6 6 6

24, 12, 12

0033F960 0033F960 0033F96C

0033F960 0033F96C

9.4 포인터 배열과 배열 포인터

1) 포인터 배열

일반 변수의 배열이 있듯이 포인터 배열이란 주소값을 저장하는 포인터를 배열 원소로 하는 배열이다. 포인터 배열도 배열 선언 시 초기값을 지정할 수 있다.

- 포인터 배열 변수선언

  자료형 *변수이름[배열크기] ;

```
// file: pointerarray.c
#define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의
#include <stdio.h>

#define SIZE 3

int main(void)
{
        //포인터 배열 변수선언
        int *pary[SIZE] = { NULL };
        int a = 10, b = 20, c = 30;

        pary[0] = &a;
        pary[1] = &b;
        pary[2] = &c;
        for (int i = 0; i < SIZE; i++)
                printf("*pary[%d] = %d₩n", i, *pary[i]);

        for (int i = 0; i < SIZE; i++)
        {
                scanf("%d", pary[i]);
                printf("%d, %d, %d₩n", a, b, c);
        }

        return 0;
}
```

2) 배열 포인터

열이 4 인 이차원 배열 ary[][4]의 주소를 저장하려면 배열 포인터 변수 ptr 을 문장 int(*ptr)[4]로 선언해야 한다.

- 괄호가 없는 int *ptr[4];는 int 형 포인터 변수 4 개를 선언하는 포인터 배열 선언 문장이다.
- int (*ptr)[4];는 열이 4 인 이차원 배열 포인터 선언 문장이다.

3) 배열 크기 연산

연산자 sizeof 를 이용한 식 (sizeof(배열이름) / sizeof(배열원소))의 결과는 배열 크기이다.

- sizeof(배열이름)은 배열의 전체 공간의 바이트 수이다.
- sizeof(배열원소)는 배열원소 하나의 바이트 수이다.

이차원 배열의 행의 수는 (sizeof(x) / sizeof(x[0]))로 계산할 수 있다.

- 여기서 sizeof(x)는 배열 전체의 바이트 수이다.
- sizeof(x[0])는 1 행의 바이트 수이며, sizeof(x[0][0])은 첫 번째 원소의 바이트 수를 나타낸다.

CHAPTER 10 함수 기초

10.1 함수 정의와 호출

1) 함수의 이해

함수는 특정한 작업을 처리하도록 작성한 프로그램 단위이다. 함수는 필요한 입력을 받아 원하는 어떤 기능을 수행한 후 결과를 반환한다. 그러므로 c 프로그램은 최소한 main()함수와 다른 함수로 구성되는 프로그램이다.

함수는 라이브러리 함수와 사용자 정의 함수로 구분할 수 있다. 사용자가 직접 개발한 함수를 사용하기 위해서는 함수선언, 함수호출, 함수 정의가 필요하다.

적절한 함수로 잘 구성된 프로그램을 모듈화 프로그램 또는 구조화된 프로그램이라 한다. 한번 정의된 함수는 여러 번 호출이 가능하므로 소스의 중복을 최소화하여 프로그램의 양을 줄이는 효과를 가져온다. 이러한 함수 중심의 프로그램 방식을 절차적 프로그래밍 방식이라 한다.

2) 함수 정의

함수정의는 함수머리와 함수몸체로 구성된다.

- 함수 정의

```
int add2(int     a, int b)
{   int sum = a + b ;
    return (sum);
}
```

- 함수머리는 반환형과 함수이름, 매개변수 목록으로 구성된다.
- 함수 머리에서 반환형은 함수 결과값의 자료형으로 간단히 반환형이라 부른다. 이 반환형에는 다양한 자료형이 올 수 있다.
- 함수 이름은 식별자의 생성규칙을 따른다.
- 함수몸체는 중괄호로 시작하여 중괄호로 종료된다.
- 함수몸체에서는 함수가 수행해야 할 문장들로 구성된다.

함수가 반환값이 없다면 반환값으로 void 를 기술한다. return 문장은 함수에서 반환값을 전달하는 목적과 함께 함수의 작업 종료를 알리는 문장이다.

3) 함수선언과 함수호출

정의된 함수를 실행하려면 프로그램 실행 중에 함수호출이 필요하다. 함수원형은 함수를 선언하는 문장이다. 함수원형 구문에서 매개변수의 변수이름은 생략할 수 있다. 함수원형은 함수선언으로 변수선언과 같이 함수를 호출하기 전에 반드시 선언되어야 한다.

- 함수원형

  반환형 함수이름(매개변수 목록)

LAB

1 에서부터 표준입력으로 받은 양의 정수까지 합을 구하는 함수 getsum()

- 함수 getsum()을 구현
- 변수 max 를 선언하여 표준입력으로 양의 정수를 입력
- 함수 getsum()을 호출하여 1 부터 max 까지의 합을 출력

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

int getsum(int); //함수원형

int main(void)
{
	int max = 0;

	printf("1에서 n까지의 합을 구할 n을 입력하시오. >> ");
	scanf("%d", &max);

	printf("1에서 %d까지의 합: %d₩n", max, getsum(max)); //함수호출

	return 0;
}

int getsum(int n)
{
	int sum = 0;
	for (int i = 1; i <= n; i++)
		sum += i;

	return sum;
}
```

10.2 함수의 매개변수 활용

1) 매개변수와 인자

함수 매개변수는 함수를 호출하는 부분에서 함수몸체로 값을 전달할 목적으로 이용된다. 함수 정의에서의 매개변수는 필요한 경우 자료형과 변수명의 목록으로 나타내며 필요 없으면 키워드 void 를 기술한다.

함수 정의에서 기술되는 매개변수 목록의 변수를 형식 매개변수라 한다. 형식매개변수는 함수 내부에서만 사용할 수 있는 변수이다.

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

int add2(int a, int b);          //int add2(int, int)도 가능
int findMax2(int, int);          //int findMax2(int a, int b)도 가능
void printMin(int, int); //int printMin(int a, int b)도 가능

int main(void)
{
        int a = 3, b = 5;

        int max = findMax2(a, b);
        printf("최대: %d₩n", max);
        printf("합: %d₩n", add2(a, b));

        //반환 값이 없는 함수호출은 일반문장처럼 사용
        printMin(2, 5);

        return 0;
}

//이하 함수 add2, findMax2, findMin2, printMin 구현
int add2(int a, int b)
{
        int sum = a + b;

        return (sum);
}

int findMax2(int a, int b)
{
        int max = a > b ? a : b;

        return max;
}

int findMin2(int x, int y)
{
        int min = x < y ? x : y;

        return (min);
}

void printMin(int a, int b)
{
        int min = a < b ? a : b;
        printf("최소: %d₩n", min);

        return;          //생략 가능
}
```

2) 배열을 매개변수로 사용

함수의 매개변수로 배열을 전달한다면 한 번에 여러 개의 변수를 전달하는 효과를 가져온다. 함수 sum()은 실수형 배열의 모든 원소의 합을 구하여 반환하는 함수이다.

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

//int sumaryf(int ary[], int SIZE);
int sumary(int *ary, int SIZE);

int main(void)
{
	int point[] = { 95, 88, 76, 54, 85, 33, 65, 78, 99, 82 };
	int *address = point;
	int aryLength = sizeof(point) / sizeof(int);

	int sum = 0;
	for (int i = 0; i < aryLength; i++)
		sum += *(point + i);
		//sum += *(point++);	//오류발생
		//sum += *(address++);	//가능

	printf("메인에서 구한 합은 %d₩n", sum);
	address = point;
	printf("함수sumary() 호출로 구한 합은 %d₩n", sumary(point, aryLength));
	printf("함수sumary() 호출로 구한 합은 %d₩n", sumary(&point[0], aryLength));
	printf("함수sumary() 호출로 구한 합은 %d₩n", sumary(address, aryLength));

	return 0;
}

//int sumary(int ary[], int SIZE)도 가능
int sumary(int *ary, int SIZE)
{
	int sum = 0;

	for (int i = 0; i < SIZE; i++)
	{
		//sum += ary[i];		//가능
		//sum += *(ary + i);	//가능
		sum += *ary++;
		//sum += *(ary++);		//가능
	}

	return sum;
}
```

[실행결과]

메인에서 구한 합은 755

함수 summary() 호출로 구한 합은 755

함수 summary() 호출로 구한 합은 755

함수 summary() 호출로 구한 합은 755

10.3 재귀와 라이브러리 함수

1) 재귀와 함수 구현

함수구현에서 자신 함수를 호출하는 함수를 재귀 함수라 한다.

다음 예제는 재귀함수 factorial 을 이용하여 1!에서 10!까지 결과를 출력한다.

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

int factorial(int); //함수원형

int main(void)
{
        for (int i = 1; i <= 10; i++)
                printf("%2d! = %d₩n", i, factorial(i));

        return 0;
}

// n! 구하는 재귀함수
int factorial(int number)
{
        if (number <= 1)
                return 1;
        else
                return (number * factorial(number - 1));
}
```

2) 난수 라이브러리 함수

특정한 나열 순서나 규칙을 가지지 않는 연속적인 임의의 수를 난수라 한다. 함수 rand()의 함수원형은 헤더파일 stdlib.h 에 정의되어 있다. 함수 rand()는 0 에서 32767 사이의 정수 중에서 하나의 정수를 임의로 반환한다. 1 에서 n 까지의 난수를 발생시키려면 함수 rand()를 이용해 수식 rand() % n + 1 를 이용한다.

함수 time(NULL)은 1970 년 1 월 1 일 이후 현재까지 경과된 시간을 초 단위로 반환하는 함수이다.

3) 수학과 문자 라이브러리 함수

수학 관련 함수를 사용하려면 헤더파일 math.h 를 삽입해야 한다.

표 10-1 수학 관련 함수

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

C 언어에는 다양한 라이브러리 함수를 제공한다. 다음과 같이 처리 작업에 따라 여러 헤더파일이 제공된다.

표 10-3 여러 라이브러리를 위한 헤더 파일

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

LAB

1 에서 100 사이의 난수를 알아 맞추는 프로그램

- 함수 time()을 이용하기 위해 헤더 파일 time.h 를 삽입한다.
- 난수에 시드를 지정하기 위해 함수 srand(long) time(NULL))을 호출한다.
- 1 에서 n 까지의 난수를 발생시키려면 함수 rand()를 이용하여 수식 rand % n+1 을 이용한다.

  [실행결과]

  1 에서 100 사이에서 한 정수가 결정되었습니다.

  이 정수는 무엇일까요? 입력해 보세요. : 90

  입력한 수보다 큽니다. 다시 입력하세요 : 75

  입력한 수보다 작습니다. 다시 입력하세요. : 85

  정답입니다.

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

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

#define MAX 100

int main(void)
{
        int guess, input;

        srand((long)time(NULL));
        guess = rand() % MAX + 1;

        printf("1에서 %d 사이에서 한 정수가 결정되었습니다.₩n", MAX);
        printf("이 정수는 무엇일까요? 입력해 보세요. : ");

        while ( scanf("%d", &input) ) {
                if (input > guess)
                        printf("입력한 수보다 작습니다. 다시 입력하세요. : ");
                else if (input < guess)
                        printf("입력한 수보다 큽니다. 다시 입력하세요. : ");
                else
                {
                        puts("정답입니다.");
                        break;
                }
        }

        return 0;
}
```

끝

읽어주셔서 감사합니다.

20164108

권경은