C 언어
포트폴리오

학부	컴퓨터공학부
학과	컴퓨터정보공학과
학번	20193431
성명	최원석

01 프로그래밍언어 개요

1.1 '프로그램'이 무엇일까?

알람 앱, 기상정보앱, 일정관리 앱, 계산기, 메모장과 같이 컴퓨터와 스마트폰에서 특정 목적의 작업을 수행하기 위한 관련 파일의 모임을 프로그램(program)이라 한다. 프로그램은 응용 프로그램(application program)을 줄인 말로 특히 스마트폰에서 사용되는 프로그램은 더 간단히 "앱" 또는 "어플"이라고 한다.

프로그램은 특정 작업을 수행하기 위하여 그 처리 방법과 순서를 기술한 명령어와 자료로 구성되어 있다. 즉, 프로그램은 컴퓨터에게 지시할 일연의 처리 작업 내용을 담고 있고, 사용자의 프로그램 조작에 따라 컴퓨터에게 적절한 명령을 지시하여 프로그램이 실행된다.

컴퓨터와 스마트폰 등의 정보기기에서 사용되는 프로그램을 만드는 사람을 프로그래머(programmer)라 한다. 일반적으로 개발자(developer)라고도 부르나, 개발자는 소프트웨어 구축을 위한 기획에서부터 분석, 설계와 개발, 구현에 이르는 모든 과정에 참여하는 사람을 말하며, 프로그래머 보다 좀 더 넓은 의미라고 이해하면 좋을 듯하다.

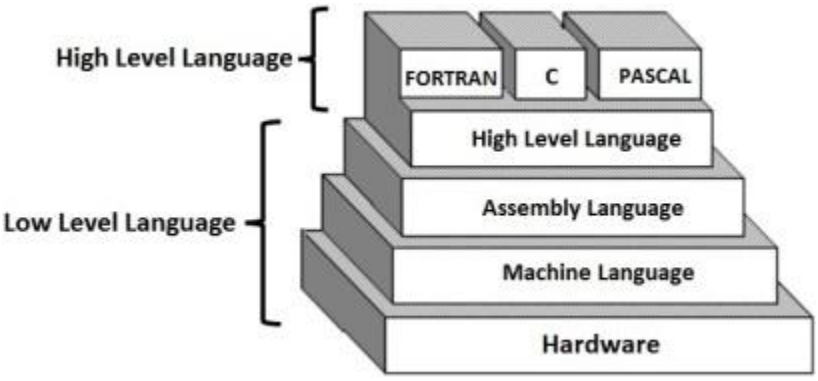
한국어, 영어, 일어, 중국어, 불어, 스페인어 등 전세계에는 매우 다양한 언어가 존재한다. 즉 우리 실생활에서 사람들과의 의사 교환을 위해 언어가 필요하듯이 사람과 컴퓨터가 서로 의사 교환을 하기 위한 언어가 프로그래밍 언어(programming language)이다. 프로그래밍 언어는 사람이 컴퓨터에게 지시할 명령어를 기술하기 위하여 만들어진 언어로, 시대와 목적에 따라 매우 다양한 프로그래밍 언어가 개발되었다. 대표적인 프로그래밍 언어로는 FORTRAN, ALGOL, BASIC, COBOL. PASCAL, C, C++, Visual Basic, Delphi, Java, Objective-C, Perl, JSP, JavaScript, Python, C#, Go 등 매우 다양하다.

1.2 언어의 계층과 번역

컴퓨터는 고철 덩어리인 하드웨어와 이 하드웨어를 작동하게 하는 소프트웨어로 구성된다. 하드웨어의 중요한 구성요소로는 중앙처리장치, 주기억장치, 보조기억장치, 입력장치, 출력장치를 들 수 있다. 소프트웨어는 컴퓨터가 수행할 작업을 지시하는 전자적 명령어들의 집합으로 구성된 프로그램을 말한다.

소프트웨어는 크게 응용 소프트웨어와 시스템 소프트웨어가 있으며 시스템 소프트웨어 중에는 크게 운영체제와 각종 유틸리티 프로그램으로 구분할 수 있다. 운영체제는 특정 CPU 에 맞게 관련된 하드웨어를 작동하게 하고 또한 응용소프트웨어를 실행해주는 소프트웨어이다. 유틸리티 프로그램은 운영체제를 돕고 컴퓨터 시스템이 원활하게 작동하도록 돕는다.

컴퓨터는 기계어를 유일하게 인식하는데, 이는 전기의 흐름을 표현하는 1 과 흐르지 않음을 의미하는 0 으로 표현된다. 이러한 기계어는 사람이 인식하기에는 난해하기 때문에 서로 의사 교환을 하려면 통역사 같은 번역기가 필요하다. 즉 사람이 프로그래밍 언어로 컴퓨터에게 명령을 내리기 위해서는 프로그래밍 언어를 기계어로 변환하는 통역사인 컴파일러가 필요하다. 컴퓨터 하드웨어에 대한 강력한 통제가 가능하다는 장점이 있으며 중앙처리장치(CPU: Central Processing Unit)에 종속(dependent)되는 언어이다.



컴퓨터의 중앙처리장(CPU)에 적합하게 만든 기계어와 어셈블리 언어를 모두 저급언어라고 볼 수 있다. 이와 반대로 컴퓨터의 CPU 에 의존하기 않고 우리 사람이 보다 쉽게 이해할 수 있도록 만들어진 언어를 고급언어라 한다.

다양한 종류의 고급언어로 작성된 프로그램은 반드시 기계어로 변환되어야 실행이 가능하다. 즉 컴파일러는 고급언어로 작성된 프로그램을 기계어 또는 목적코드로 바꿔주는 프로그램이다.

1.3 왜 c 언어를 배워야 할까?

C 언어는 1972 년 데니스 리치가 개발한 프로그래밍 언어이다. 당시 미국전신전화국의 벨 연구소에 근무하던 데니스 리치는 시스템 PDP-11 에서 운용되는 운영체제인 유닉스를 개발하기 위해 C 언어를 개발하였다. C 언어는 어셈블리 언어 정도의 속도를 내며, 좀 더 쉽고, 서로 다른 CPU 에서도 작동되도록 개발되었다.

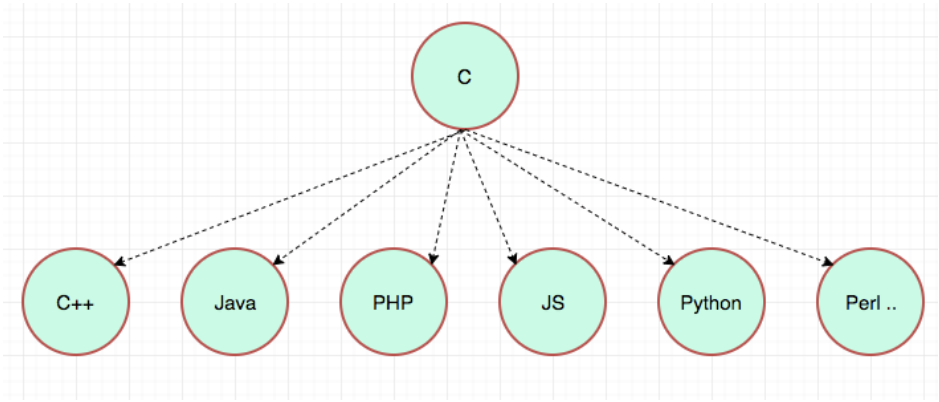
1983 년 미국전신전화국에 근무하는 얀 스트로스트럽은 C 언어에 객체지향프로그래밍 개념을 확장시킨 프로그래밍 언어 C++를 개발하였다. 1995 년에는 중대형 컴퓨터 시스템 회사로 유명한 선 마이크로시스템즈사는 C++언어를 발전시켜 인터넷에 적합한 언어인 JAVA 를 발표하였다.

C 언어는 함수 중심으로 구현되는 절차지향 언어(procedural language)이다. 절차지향 언어란 시간의 흐름에 따라 정해진 절차를 실행한다는 의미로 C 언어는 문제의 해결 순서와 절차의 표현과 해결이 쉽도록 설계된 프로그램 언어이다.

C 언어는 간결하고 효율적인 언어이다. 시스템의 세세한 부분까지 제어할 수 있도록, 포인터(pointer)와 메모리관리(memory management) 기능을 갖고 있다. C 언어로 작성된 프로그램은 크기가 작으며, 메모리도 적게 효율적으로 사용하여 실행 속도가 빠르다.

C 언어는 이식성(portability)이 좋다. 즉 C 로 작성된 소스는 별다른 수정 없이 다양한 운영체제의 여러 플랫폼에서 제공되는 컴파일러로 번역해 실행될 수 있다.

C 언어의 단점은 다소 배우기 어렵다는 것이다. C 언어의 문법이 상대적으로 간결한 대신 많은 내용을 함축하고 있으며, 비트(bit)와 포인터(pointer)의 개념, 메모리 할당과 해제 등의 관리로 실제 C 언어는 조금 어려운 것이 사실이다. 그러나 C 언어는 다른 프로그래밍 언어에도 많은 영향을 끼친 언어이므로 한번 익히면 다른 프로그래밍 언어 습득에도 많은 도움을 준다.



1.4 프로그래밍의 자료 표현

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

컴퓨터 메모리의 저장 단위 또는 정보 처리 단위 중에서 가장 작은 기본 정보 단위(basic unit of information)가 비트(bit)이다. 비트는 BInary digiT 의 합성어다. 비트가 연속적으로 8 개가 모인 정보 단위를 바이트(byte)라 한다. 1 바이트는 8 비트를 조합하므로 2^8=256 가지의 정보 종류를 저장할 수 있다.

참(true)과 거짓(false)을 의미하는 두 가지 정보를 논리값이라 한다. 하나의 비트 정보도 0 과 1 이므로 이를 각각 거짓과 참으로 표현할 수 있다.

컴퓨터 장치를 개발한 회사들이 서로 달라서 약속한 정보가 서로 다르다면 제대로 문자를 전송할 수 없다는 문제점이 생기게 된다. 예를 들어, K 라는 회사는 'A'문자 값을 65 으로 정의하고 S 라는 회사는 'A'값을 75 로 정의해서 제품을 생산했다면 K 회사에서 만든 제품과 S 회사에서 만든 제품은 서로 문자를 주고받을 수 없다.

이러한 문제를 해결하기 위해서 여러 표준이 나왔는데 그 중 개인용 컴퓨터에서 가장 많이 사용하는 표준이 바로 아스키 코드이다. 아스키(ASCII: American Standard Code for Information Interchange) 코드는 1976 년에 표준으로 제정되어 1986 년에 마지막으로 개정되었다.

유니코드(Unicode)는 전 세계 모든 언어를 하나의 코드 체계 안으로 통합하기 위하여 만들어진 코드이다. 즉 유니코드는 전 세계의 모든 문자를 컴퓨터에서 일관되게 표현하고 다룰 수 있도록 설계된 산업 표준이며, 유니코드 협회가 제정하여 1991 년 버전 1.0 이 발표되었다. 아스키 코드는 영어 문자 기반의 코드 체계이기 때문에 동양권의 2 바이트 문자 체계를 수용하기에는 다소 무리가 있는 시스템이다. 기존의 아스키에서 사용되었던 8 비트 체계에서 벗어나, 전 세계의 문자를 모두 표현하기 위해 2 바이트 즉, 16 비트로 확장된 코드 체계가 유니코드이다.

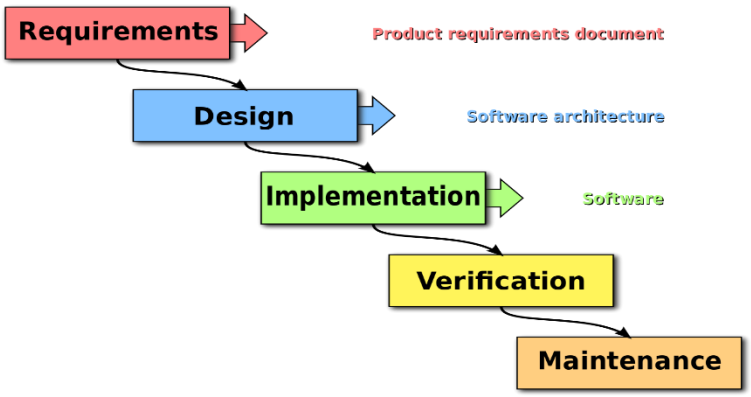
1980 년 중반부터 논의가 시작되어 1991 년 버전 1.0 이 제정되었으며, 1996 년 65,536 자의 코드영역을 언어학적으로 분류하였으며, 한글 완성자 11,172 자의 한글과 중국, 일본을 포함해 시계 유수의 언어 문자를 배열해 만든 유니코드는 국제표준화기구에 상정 확정되었다.

1.5 소프트웨어 개발

소프트웨어는 보통 '프로그램'이라고 부르는 것 외에도 데이터와 문서 까지를 포함하는 포괄적인 개념이다. 컴퓨터 프로그램은 특정한 업무를 수행하기 위한 정교한 알고리즘의 집합이라고 간주할 수 있다. 알고리즘이란 어떠한 문제를 해경하기 위한 절차나 방법으로 명확히 정의된 유한 개의 규칙과 절차의 모임이다.

소프트웨어의 개발 과정을 연구하는 분야를 소프트웨어 공학(software engineering)이라 한다. 소프트웨어 공학이란 공학적 원리에 의하여 소프트웨어를 개발하는 학문으로, 소프트웨어 개발과정인 분석, 설계, 개발, 검증, 유지보수 등 개발수명주기 전반에 걸친 계획, 개발, 검사, 보수, 관리, 방법론 등을 연구하는 분야이다.

소프트웨어의 개발 단계를 구체적으로 살펴보면 다음과 같다.



1. 요구사항 분석 단계: 시스템을 사용할 사용자의 요구사항을 파악하여 분석하는 단계이다.
2. 설계 단계: 프로그래머가 알고리즘을 이용하여 소프트웨어를 설계한다.
3. 구현 단계: 흐름도 또는 의사코드를 컴퓨터가 이해할 수 있는 자바나 C 와 같은 특정한 프로그래밍 언어로 개발한다.
4. 검증 단계: 프로그램의 소프트웨어 요구사항에 얼마나 부합하는 프로그램이 안정적으로 작동하는지를 검사한다.
5. 유지보수 단계: 관리를 쉽게 하기 위해서는 체계적인 문서화가 필요하며, 필요에 따라 프로그램을 적절하게 변경시켜야 하는 등 유지보수 작업을 한다.

1.6 다양한 '프로그래밍 언어'

- 포트란: IBM 704 시스템에서 과학과 공학 및 수학적 문제들을 해결하기 위해 고안된 프로그래밍 언어로 널리 보급된 최초의 고급언어이다. 포트란은 가장 오래된 언어지만 언어 구조가 단순해 놀라운 생명력을 갖추고 있어 지금도 과학 계산 분야 등에서는 많이 사용되고 있다. 기본적인 수치처리와 적분, 유체역학, 복소수, 방정식 등 계산을 위주로 만들어졌기 때문에 매우 단순하고 간결하여 배우기가 용이하다.

- 코볼: 1950 년 말, 미국 국방성에서는 1,000 여개의 다양한 컴퓨터를 사용하였고, 수 많은 상품정보와 급여 정보, 그리고 원가 계산 등의 상업용 처리에는 포트란이 적합하지 않았다. 또한 사무용 처리로 사용하던 프로그래밍 언어가 개발업체마다 달라서 여러 문제가 있었다. 이러한 것을 인식한 미국 국방성에서는 사무처리 언어를 통일을 위해 사무처리에 대한 언어발달 모형이 제시되었고, 컴퓨터 언어에 관한 특별 회의를 위한 데이터 시스템 언어 협회(CODASYL)가 설립되었다. 학계 위원과 허니웰, GE, 뷰로우, RCA, IBM 등의 업체가 참여한 협회 CODASYL 이 1960 년, 사무처리에 적합한 프로그래밍 언어로 개발한 것이 코볼이다.

- C++: 객체지향 프로그래밍을 지원하기 위해 C 언어가 가지는 장점을 그대로 계승하면서 객체의 상속성 등의 개념을 추가한 효과적인 언어이다. C 언어의 확장이라 볼 수 있으므로 기존의 C 언어로 개발된 모든 프로그램을 수정 없이 사용할 수 있다.

- 파이썬: 현재 미국의 대학에서 컴퓨터 기초 과목으로 가장 많이 가르치는 프로그래밍 언어 중 하나이다. 무료이며, 간단하여 쉽고 빠르게 개발할 수 있다.

- 자바: 1995 년에 선 마이크로 시스템즈사에서 개발한 C++를 기반으로 한 객체 지향 프로그래밍 언어이다. 자바의 개발도구인 JDK 가 있으며, 현재 자바는 오라클 기술이 되었으며, 자바개발환경인 JDK 는 현재까지 계속 발표되고 있다.

- 스크래치: 2007 년 MIT 대학의 미디어랩에서 개발한 비주얼 프로그래밍 개발도구이다. 프로그래밍에 입문한 학생들을 대상으로 컴퓨터 프로그래밍의 개념을 이해할 수 있도록 도와주는 교육용 프로그래밍 언어이다.

02 C 프로그래밍 첫걸음

2.1 프로그래밍 구현과정과 통합개발 환경

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

- 프로그램구상과 소스편집: 프로그램을 개발하기 위해 가장 먼저 해야 할 일은 소스파일을 어떻게 작성해야 할 지 그 내용에 대해 생각하는 것이다.
- 컴파일: 고급 프로그래밍 언어에서 기계어를 만들어 내는 과정을 컴파일 과정이라 하며, '컴파일한다' 라고 한다. 컴파일러에 의해 처리되기 전의 프로그램을 소스코드라면 컴파일러에 의해 기계어로 번역된 프로그램은 목적코드라고 한다.
- 링크와 실행: 링커는 하나 이상의 목적파일을 하나의 실행파일로 만들어 주는 프로그램이다. 자주 사용하는 프로그램들은 프로그램을 작성할 때, 프로그래머마다 새로 작성할 필요 없이 개발환경에서 미리 만들어 컴파일해 저장해 놓는데, 이 모듈을 라이브러리라 한다. 비주얼 스튜디오에서는 컴파일과 링크 과정을 하나로 합쳐 빌드(build)라 한다.

프로그램 개발 과정에서 나타나는 모든 문제를 오류 또는 에러라고 한다. 오류는 그 발생 시점에 따라 컴파일 오류와 링크 오류, 실행 오류로 구분할 수 있다.

- 컴파일 오류: 개발환경에서 오류 내용과 위치를 어느 정도 알려주므로 오류를 찾아 수정하기가 비교적 쉽다.
- 링크 오류: main() 함수 이름이나 라이브러리 함수 이름을 잘못 기술하여 발생하는 경우가 대부분이다.
- 실행 오류: 링크까지 성공했는데 실행하면서 오류가 발생한 경우. 간혹 문법적인 문제가 영향을 미치는 경우도 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. 다양한 오류를 찾아 소스를 수정하여 다시 컴파일, 링크, 실행하는 과정을 디버깅(debugging)이라 하며, 이를 도와주는 프로그램을 디버거(debugger)라 한다.

2.3 c 프로그래밍의 이해와 디버깅 과정

함수 용어

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

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 종료된다.

코드

`#include <stdio.h>`
`int main(void){`
`  printf("Hello World!\n");`
`  return 0;`
`}`

결과
Hello World!

라이브러리 함수 printf()를 사용하려면 첫 줄에 #include<stdio.h>를 넣어야 한다. #include 는 바로 뒤에 기술하는 헤더 파일 stdio.h 를 삽입하라는 명령어이다.

함수 printf() 는 원하는 문자열을 괄호 (“원하는 문자열”) 사이에 기술하면 그 인자를 현재 줄의 출력 위치에 출력하는 함수이다. 함수 puts()는 원하는 문자열을 괄호 (“원하는 문자열”) 사이에 기술하면 그 인자를 현재 위치에 출력한 후 다음 첫 열로 이동하여 출력을 기다리는 함수이다.

소스 코딩을 하다 보면 ;나 “”, <>, (), {} 등의 구두 문자를 빠뜨려서 발생하는 컴파일 에러는 흔하게 발생하게 된다. 대부분의 이러한 어류는 통합개발환경 IDE 가 잡아줄 수 있다.

만들어진 부품을 조립하는 링크과정에서 발생하는 오류를 ‘링크 오류’라고 한다. 대표적인 링크 오류는 라이브러리 함수인 printf()의 철자를 잘못 기술하는 것이다. 구동 함수인 main()을 mein()등으로 잘못 기술해도 링크 오류가 발생한다.

03 자료형과 변수

3.1 프로그래밍 기초

C 프로그램은 하나 이상의 여러 함수가 모여 한 프로그램으로 구성된다는 것을 알 수 있다. 개발환경인 비주얼 스튜디오에서 솔루션은 여러 개의 프로젝트를 가지며, 다시 프로젝트는 여러 소스파일을 포함한 여러 자원으로 구성된다. 한 프로젝트는 단 하나의 함수 main()과 다른 여러 함수로 구현되며, 최종적으로 프로젝트 이름으로 하나의 실행 파일이 만들어진다.

프로그래밍 언어에서는 문법적으로 고유한 의미를 갖는 예약된 단어가 있다. 이러한 예약어를 키워드라고 한다. 키워드는 C 프로그램 자체에서 예약된 단어들이다. 프로그래머가 자기 마음대로 정의해서 사용하는 단어들을 식별자라고 한다. 이러한 식별자에는 아래와 같은 사용규칙들이 있다.

- C 프로그램에서의 예약자인 키워드와 비교하여 철자라든지, 대문자, 소문자 등 무엇이라도 달라야 한다. 대표적인 식별자로는 미리 정의하여 사용하는 변수이름 age, year 등과 함수이름으로 puts, main, printf 등이 있다.
- 식별자는 영문자(대소문자 알파벳), 숫자(0~9), 밑줄(_)로 구성되며, 식별자의 첫 문자로 숫자가 나올 수 없다.
- 키워드는 식별자로 이용할 수 없다.
- 식별자는 대소문자를 모두 구별한다. 예를 들어, 변수 Count, count. COUNT 는 모두 다른 변수이다.
- 식별자의 중간에 공백 문자를 들어갈 수 없다.

프로그래밍 언어에서 컴퓨터에게 명령을 내리는 최소 단위를 문장(statement)이라 하며, 문장은 마지막에 세미콜론(;)으로 종료된다.

여러 개의 문장을 묶으면 블록(block)이라고 하며 { 문장 1, 문장 2 } 처럼 중괄호로 열고 닫는다.

주석(//) // 이후부터 그 줄의 마지막까지 주석으로 인식한다. 블록주석(/* */) 여러 줄에 걸쳐 설명을 사용할 때 이용한다.

3.2 자료형과 변수 선언

자료형은 프로그래밍 언어에서 자료를 식별하는 종류를 말한다.

변수는 자료값을 저장하는 공간이며, 고유한 이름이 붙여진다. 선언된 자료형에 따라 변수의 저장공간 크기와 저장되는 자료값의 종류가 결정된다.

변수선언은 컴파일러에게 프로그램에서 사용할 저장 공간인 변수를 알리는 역할이며, 프로그래머 자신에게도 선언한 변수를 사용하겠다는 약속의 의미가 있다.

코드

```c
#include <stdio.h>
int main(void) {
	int number;
	number = 20193431;
	printf("학번 : %d\n", number);
}
```

결과

```
학번 : 20193431
```

원하는 자료값을 선언된 변수에 저장하기 위해서는 대입연산자 표시인 '='를 사용한다. 대입연산자 '='는 오른쪽의 값을 이미 선언된 변수에 저장한다는 의미이다. 이 대입연산이 있는 문장은 대입문이라 한다.

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

변수에서 주요 정보인 변수이름, 변수의 자료형, 변수 저장 값을 변수의 3 요소라 한다. 죽 변수 선언 이후 저장 값이 대입되면 변수의 3 요소가 결정된다. 변수 선언 이후 변수 3 요소 중에서 변수 이름과 변수형은 바뀌지 않으나 대입 문장에 의해 변수 저장 값은 계속 바뀔 수 있다. 저장 값이 계속 바뀔 수 있으므로 변수라 하는 것이다.

변수의 의미는 저장공간 자체와 저장공간에 저장된 값으로 나눌 수 있다. 대입 연산자 =의 왼쪽에 위치한 변수는 저장공간 자체의 사용을 의미한다. 그러나 대입 연산자 -의 오른쪽에 위치한 변수는 저장 값의 사용을 의미한다.

3.3 기본 자료형

코드

```c
#include <stdio.h>
int main(void) {
	int ch = 126;
	printf("%d\n", ch);		//십진 코드값 출력
	printf("%c\n", ch);		//문자 출력
	printf("%c\n", '\176');		//문자 출력
	printf("%c\n", '\x7e');		//문자 출력
	return 0;
}
```

결과

```
126
~
~
~
```

C 의 자료형은 기본형, 유도형, 사용자정의형 등으로 나눌 수 있으며, 기본형은 기본이 되는 자료형으로 다시 정수형, 부동소수형, 문자형, 무치형으로 나뉜다. 유도형은 기본형에서 나온 자료형으로 배열, 포인터, 함수 등으로 구성된다. 사용자 정의형은 기본형과 유도형을 이용하여 프로그래머가 다시 만드는 자료형으로 열거형(enumeration), 구조체(structure), 공용체 (union) 등이 있다.

정수형의 기본 키워드는 int 이다. int 형으로 선언된 변수에는 365, 1024, 030, 0xF3 과 같이 십진수, 팔진수, 십육진수의 정수가 다양하게 저장될 수 있다.

부동소수형은 3.14, 3.26567 과 같이 실수를 표현하는 자료형이다. 부동소수형을 나타내는 키워드는 float, double, long double 세 가지이다.

문자형 char 는 영단어 character(문자)의 약자이며, 크기는 1 바이트이다.

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

자료형의 범주에서 벗어난 값을 저장하면 오버플로(overflow)또는 언더플로(underflow)가 발생한다. 정수형 자료형에서 최댓값+1 은 오버플로로 인해 최솟값이 된다. 마찬가지로 최솟값-1 은 최댓값이 된다. 이러한 특징을 정수의 순환이라고 한다.

3.4 상수 표현방법

부동소수형 최대 최소 매크로 상수 출력

```c
#include <stdio.h>
#include <float.h>
int main(void) {
          printf("float 범위: %e %e\n", FLT_MIN, FLT_MAX);
          printf("double 범위: %e %e\n", FLT_MIN, FLT_MAX);
          printf("long double 범위: %e %e\n", LDBL_MIN, LDBL_MAX);
          return 0;
}
```

결과

```
float 범위: 1.175494e-38 3.402823e+38

double 범위: 1.175494e-38 3.402823e+38

long double 범위: 2.225074e-308 1.797693e+308
```

상수는 이름 없이 있는 그대로 표현한 자료값이나 이름이 있으나 정해진 하나의 값만으로 사용되는 자료값을 말한다.

문자 상수는 문자 하나의 앞 뒤에 작은따옴표를 넣어 표현한다.

정수형 상수는 int, unsinged int, long, unsigned long, long long, unsigned long long 등의 자료형으로 나뉜다. 일반 정수는 int 유형이며, 정수 뒤에 L 을 붙이면 long int 를 나타낸다.

숫자 0 을 정수 앞에 높으면 팔진수로 인식하며, 0x 를 숫자 앞에 높으면 십육진수로 인식한다.

변수선언 시 자료형 또는 변수 앞에 키워드 const 가 놓이면 이 변수는 심볼릭 상수가 된다. 상수는 변수선언시 반드시 초기값을 저장해야 한다.

열거형은 키워드 enum 을 사용하여 정수형 상수 목록 집합을 정의하는 자료형이다. 열거형 상수에서 목록 첫 상수의 기본값이 0 이며 다음부터 1 씩 증가하는 방식으로 상수 값이 자동으로 부여된다.

전처리기 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.

04 전처리와 입출력

4.1 전처리

C 언어는 컴파일러가 컴파일 하기 전에 전처리기의 전처리 과정이 필요하다. 이후 컴파일러는 전처리기가 생성한 소스를 컴파일한다.

- 전처리 과정에서 처리되는 문장을 전처리 지시자라 한다.
- #include #define 과 같은 전처리 지시자는 항상 #으로 시작하고, 마지막에 세미콜론(;) 이 없는 등 일반 C 언어 문장과는 구별된다.
- 조건 지시자로 #if, #elif, #else, #endif, #ifdef, #ifndef, #undef 등이 있다.

#include: 헤더파일을 삽입하는 지시자이다. 헤더파일은 자료형의 재정의(typedef), 함수원형(prototype) 정의 등과 같은 문장이 있는 텍스트 파일이다. 대표적인 헤더파일인 stdio.h 는 printf(), scanf(), putchar(), getchar() 등과 같은 입출력 함수를 위한 함수원형 등이 정의된 헤더파일이다.

#define: 매크로 상수를 정의하는 지시자이다. 다른 일반 변수와 구분하기 위해 #define 에 의한 심볼릭 상수도 주로 대문자 이름으로 정의하는데, 이를 매크로 상수라고 부른다. 전처리기는 소스에서 정의된 매크로 상수를 모두 #define 지시자에서 정의된 문자열로 대체시킨다. 매크로는 이미 정의된 매크로를 다시 사용할 수 있다.

문자열 출력을 위한 매크로 정의

```c
#include <stdio.h>
#define myprint(x) printf(x);\
puts("");
int main(void) {
        myprint("매크로로 출력하기");
        printf("출력함수로 출력하기\n");
}
```

결과

```
매크로로 출력하기
출력함수로 출력하기
```

4.2 출력 함수 printf()

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

정수와 실수, 문자와 문자열의 출력

```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);
}
```

결과

성별: M
이름: 안 병훈
나이: 20
몸무게: 62.49
평균평점(GPA): 3.880

형식지정자	값	결과	형식지정자	값	결과
%d	1234	1234	%-#6o	037	037___
%6i	1234	__1234	%#-6o	037	037___
%+6i	1234	_+1234	%05x	0x1f	0001f
%+06i	1234	+01234	%0#6x	0x1f	0x001f
%6o	037	____37	%-05#5x	0x1f	0x1f_
%-6o	037	37	%#6x	0x1f	__0x1f
%c	‘Q’	Q	%f	3.141592f	3.141592
%10f	3.141592f	___3.141592	%10.4f	3.141592f	_____3.1416
%+10.4f	3.141592f	____+3.1416	%-10.4f	3.141592f	3.1416_____

4.2 출력 함수 scanf()

십진수, 팔진수, 십육진수인 세 정수를 입력 받아 적절히 출력

```
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
int main(void) {
        int dec = 30, oct = 012, hex = 0x5E;
        printf("세 개의 정수를 각각 다음과 같이 입력하세요. ");
        printf("십진수 - 팔진수 - 십육진수\n");
        scanf("%d - %o - %x", &dec, &oct, &hex);
        printf("\n 입력된 수는 다음과 같습니다.\n");
        printf("%d - %o - %x\n", dec, oct, hex);
        printf("%d - %d - %d9\n", dec, oct, hex);
}
```

결과

```
세 개의 정수를 각각 다음과 같이 입력하세요. 십진수 - 팔진수 - 십육진수
10 - 65 - f3

입력된 수는 다음과 같습니다.
10 - 65 - f3
10 - 53 - 2439
```

printf()가 대표적인 출력 함수라면, 함수 scanf()는 대표적인 입력 함수이고 %s 와 %d 같은 동일한 형식 지정자를 사용한다.

- 함수 printf() 처럼 함수 scanf()에 첫 번째 인자는 형식문자열이라 하며 형식 지정자와 일반 문자로 구성된다.
- 형식지정자는 %d, %c, %lf, %f 와 같이 %로 시작한다.
- 함수 scanf()에서 두 번째 인자부터는 키보드 입력 값이 복사 저장되는 입력변수 목록에서 변수 이름 앞에 반드시 주소 연산자 &를 붙여 나열한다.
- 함수 scanf()의 반환 유형은 int 로, 표준입력으로 변수에 저장된 입력 개수를 반환한다.
- &는 주소연산자로 뒤에 표시된 피연산자인 변수 주소 값이 연산 값으로, scanf()의 입력 변수 목록에는 키보드에 입력 값이 저장되는 변수를 찾는다는 의미에서 반드시 변수의 주소연산식 '&변수이름'이 인자로 사용되어야 한다.
- 만일 주소연산이 아닌 변수명으로 기술하면 저장될 주소를 찾지 모해 오류가 발생한다.

05 연산자

5.1 연산식과 다양한 연산자

표준 입력된 두 실수의 산술연산 출력

```c
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
int main(void) {
	double a = 0, b = 0;
	printf("산술연산을 수행할 두 실수를 입력하세요\n");
	scanf("%lf, %lf", &a, &b);
	printf("%8.2f + %8.2f ==> %8.2f\n", a, b, a + b);
	printf("%8.2f - %8.2f ==> %8.2f\n", a, b, a - b);
	printf("%8.2f * %8.2f ==> %8.2f\n", a, b, a * b);
	printf("%8.2f / %8.2f ==> %8.2f\n", a, b, a / b);
	return 0;
}
```

결과

```
산술연산을 수행할 두 실수를 입력하세요
54.987, 4.87654
   54.99 +     4.88 ==>    59.86
   54.99 -     4.88 ==>    50.11
   54.99 *     4.88 ==>   268.15
   54.99 /     4.88 ==>    11.28
```

산술연산자는 +, -, *, /, %로 각각 더하기, 빼기, 곱하기, 나누기, 나머지 연산자이다. 나머지 연산식 a% b 의 결과는 a 를 b 로 나눈 나머지 값이다. 즉 나머지 연산식 10 % 3 의 결과는 1 이다. 나머지 연산식 %의 피연산자는 반드시 정수이어야 한다. 피연산자가 실수이면 오류가 발생한다.

부호연산자 +,-는 연산식 +3, -4.5, -a 와 같이 수 또는 변수의 부호로 표기하는 연산자 +, -는 단항연산자이다.즉 -a 는 a 의 부호가 바뀐 값이 결과 값이다.

대입연산자 =는 연산자 오른쪽의 연산 값을 변수에 저장하는 연산자이다. 대입연산자의 왼쪽 부분에는 반드시 하나의 변수만이 올 수 있다. 이 하나의 변수를 왼쪽을 의미하는 left 단어에서 l-value 라 하며 오른쪽에 위치한 연식의 값을 오른쪽을 의미하는 right 단에서 r-value 라한다.

대입 연신식 a=a+b 는 중복된 a 를 생략하고 간결하게 a+=b 로 쓸 수 있다. 마찬가지로 a=a-b 는 간결하게 a-=b 로 쓸 수 있다. 이와 같이 산술연산자와 대입연산자를 이어 붙인 연산자 +=, -=, *=, /=, %=을 축약 대입연산자라 한다. 즉 a+=2 는 a=a+2 의 대입연산을 의미한다.

5.2 관계와 논리, 조건과 비트연산자

표준 입력으로 받은 두 정수의 비트 연산 수행 출력

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int main(void) {
	int a = 0, b = 0;
	printf("비트 연산이 가능한 두 정수를 입력하세요.\n");
	scanf("%d %d", &a, &b);
	printf("%3d & %3d ==> %3d\n", a, b, a & b);
	printf("%3d | %3d ==> %3d\n", a, b, a | b);
	printf("%3d ^ %3d ==> %3d\n", a, b, a ^ b);
	printf(" ~%3d ==> %3d\n", a, ~a);
	printf("%3d >> %3d ==> %3d\n", a, b, a >> b);
	printf("%3d << %3d ==> %3d\n", a, b, a << b);
}
```

결과

```
비트 연산이 가능한 두 정수를 입력하세요.
40 3
 40 &   3 ==>   0
 40 |   3 ==>  43
 40 ^   3 ==>  43
 ~ 40 ==> -41
 40 >>   3 ==>   5
 40 <<   3 ==> 320
```

관계연산자는 두 피연산자의 크기를 비교하기 위한 연산자이다.

- 관계연산자 !=, >=, <=는 연산 기호의 순서가 명확해야 한다.
- 또한 관계연산자 ==는 피연산자 두 값이 같은지를 알아보는 연산자로 대입연산자 =와 혼동하지 않도록 주의해야한다.

논리연산자 &&, ||, ! 은 각각 and, or, not 의 논리연산을 의미한다. 논리연산자 &&과 ||는 피연산자 두 개 중에서 왼쪽 피연산자만으로 논리연산 결과가 결정한다면 오른쪽 피연산자는 평가하지 않는다.

조건 연산자 (a? b:c)는 조건에 따라 주어진 피연산자가 결과 값이 되는 삼항연산자이다.

비트 논리 연산자는 피연산자 값을 비트 단위로 논리 연산을 수행하는 연산자로, &, |, ^, ~가 있다.

비트 이동 연산자 >>, <<는 연산자의 방향인 왼쪽이나 오른쪽으로, 비트 단위로 줄줄이 이동시키는 연산자이다.

5.3 형변환 연산자와 연산자 우선순위

섭씨 온도를 화씨 온도로 출력

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int main(void) {
        double fahrenheit, celsius;
        printf("변환할 섭씨온도를 입력하세요. >> ");
        scanf("%lf", &celsius);
        fahrenheit = (9.0 / 5.0) * celsius + 32.0;
        printf("\n 입력된 %.2f 도는 화씨온도로 %.2f 도 입니다.\n", celsius, fahrenheit);
        return 0;
}
```

결과

```
변환할 섭씨온도를 입력하세요. >> 34.765879

입력된 34.77 도는 화씨온도로 94.58 도 입니다.
```

자료형 char 와 int 는 각각 문자와, 정수를 표현하고 각각 1 바이트와 4 바이트로 크기도 다르다. 이러한 char 형과 int 형 간과 같이 필요에 따라 자료의 표현방식을 바꾸는 것을 자료형 변환이라 한다. 자료형 변환은 크기 자료형의 범주 변화에 따른 구분으로 올림변환과 내림변환으로 나눌 수 있다.

- 올림변환: 작은 범주의 자료형(int)에서 보다 큰 범주인(double)으로의 형변환 방식
- 내림변환: 큰 범주의 자료형(double)에서 보다 작은 범주인(int)으로의 형변환 방식

또 다른 자료형 변환 구분 방식으로 명시적 형변환과 묵시적 형변환으로 나눈다.

- 명시적(강제) 형변환): 소스에서 직접 형변환 연산자를 사용하는 방식
- 묵시적(자동 형변환): 컴파일러가 알아서 자동으로 수행하는 방식

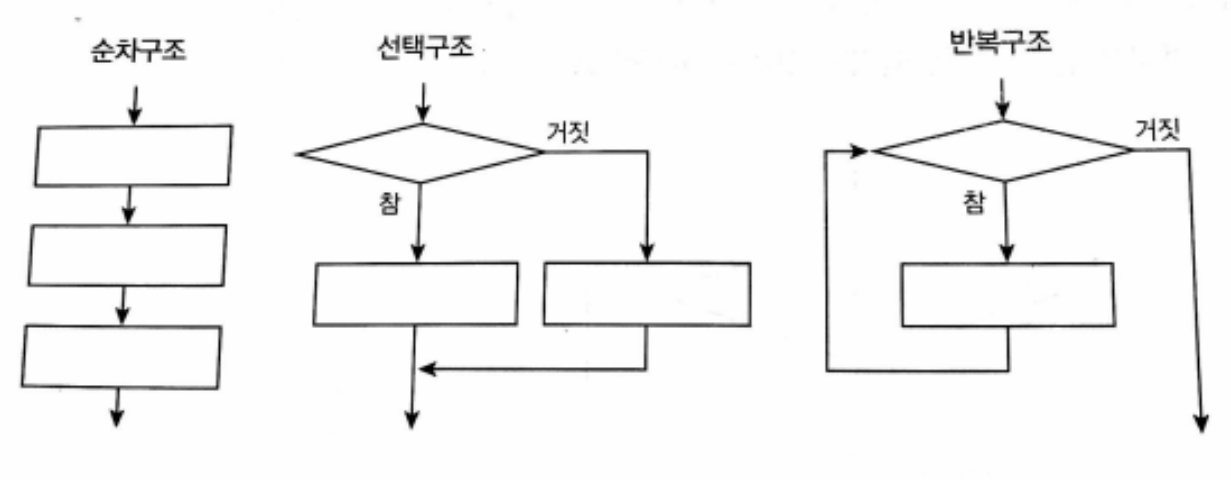
연산자 sizeof 는 연산값 또는 자료형의 저장장소의 크기를 구하는 연산자이다. 연산자 sizeof 의 결과값은 바이트 단위의 정수이다.

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

06 조건

6.1 제어문 개요

C 언어에서 제공하는 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. 우리의 인생도 중요한 선택의 기로에서 여러 길 중에서 하나를 선택하듯이 문제를 해결하는 프로그램에서 여러 개의 사항 중에 하나를 선택하는 조건선택이 자주 활용된다.

반복 또는 순환 구문이란 정해진 횟수 또는 조건을 만족하면 정해진 몇 개의 문장을 여러 번 실행하는 구문이다. 우리 삶도 어떻게 보면 동일한 일을 반복하는 경우가 많듯이 실제 프로그램에서도 반복 문장의 비중은 상당히 높아 매우 중요한 구문이다.

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

6.2 조건에 따른 선택 if 문

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

```
#define _CRT_SECURE_NO_WARNINGS
#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);
}
```

결과

```
두 실수를 입력: 32.765 3.987
연산 결과: 8.22
```

문장 if 은 조건에 따른 선택을 지원하는 구문이다. 가장 간단한 if 문의 형태는 if(cond) stmt;이다.

if 문에서 조건식 cond 가 0 이 아니면 stmt 를 실행하고, 0 이면 stmt 를 실행하지 않는다. 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 하도록 한다.

if 문은 조건이 만족되면 특정한 문장을 실행하는 구문이다. 반대로 조건이 만족되지 않은 경우에는 실행할 문장이 있다면 else 를 사용한다. 조건문 if 에서 키워드 else 를 사용하여 조건 연산값이 0 이면 else 이후의 문장을 실행하는 구문을 만들 수 있다. 조건문 if (cond) stmt1; else stmt2;는 조건 cond 를 만족하면 stmt1 을 실행하고, 조건 cond 를 만족하지 않으면 stmt2 를 실행하는 문장이다.

6.3 다양한 선택 switch 문

표준입력으로 받은 정수에 대응하는 열거 상수로 switch 문 활용

```
#define _CRT_SECURE_NO_WARNINGS
#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");
		break;
	}
}
```

결과

```
세 정수(R[0], G[1], B[2]) 중의 하나를 입력: 0
Red
```

다중선택 구문인 switch 문을 사용하면, 문장 if else 가 여러 번 계속 반복되는 구문을 좀 더 간략하게 구현할 수 있다. 특히 if 의 조건식이 정수와 등호식이라면 간편한 swtich 문의 사용이 가능하다. switch 문은 주어진 연산식이 문자형 또는 정수형이라면 그 값에 따라 case 의 상수값과 일치하는 부분의 문장들을 수행하는 선택 구문이다.

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. 이러한 break 문의 특징을 살려 case 4: case 5:로 여러 개의 case 를 나열한 후 필요한 곳에만 break 문을 배치한다면 서로 다른 여러 값에 대한 동일한 기능을 수행할 수 있다. 그러나 case 4, 5 와 같은 나열은 문법오류가 발생하니 주의하자.

07 반복

7.1 반복 개요와 while 문

0 부터 20 까지 3 의 배수 출력

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#define MAX 20
int main(void) {
	int n = 0;
	while (n <= MAX) {
		printf("%4d", n);
		n += 3;
	}
	return 0;
}
```

결과

```
0   3   6   9  12  15  18
```

C 언어는 while, do while, for 세가지 종류의 반복 구문을 제공한다. 반복은 순환 (loop)이라고도 부르며, 먼저 반복조건을 검사하여 반복을 수행하는 while 구문, 제일 나중에 반복 조건 검사하여 반복을 수행하는 do while 구문, 초기화와 반복조건, 그리고 증감연산의 세부분으로 나누어 일정한 횟수의 반복에 적합한 for 구문으로 나눌 수 있다. 반복조건을 만족하면 일정하게 반복되는 부분을 반복몸체라고 한다.

반복문 do while 은 조건식이 반복몸체 뒤에 위치하므로 처음에 조건을 검사할 수 없다. 따라서 무조건 한 번 실행한 후 조건을 검사하고 이때 조건식이 참이면 반복을 더 실행한다.

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

- 이 반복은 cond 가 0(거짓)이 될 때까지 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.

7.2 do while 문과 for 문

구구단을 위한 준비 출력

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#define MAX 9
int main(void) {
          printf("=== 구구단 출력 ===\n");
          for (int i = 2; i <= MAX; i++) {
                    printf("%6d 단 출력\n", i);
          }
          return 0;
}
```

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
```

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

- 초기화를 위한 init 를 실행한다. 이 init 는 단 한번만 수행된다.
- 반복 조건 검사 cond 를 평가해 0 이 아닌 결과값이면 반복문의 몸체에 해당하는 문장 stmt 를 실행한다. 그러나 조건검사 cond 가 결과값이 0 이면 for 문을 종료하고 다음 문장을 실행한다.
- 반복몸체인 stmt 를 실행한 후 증감연산 inc 를 실행한다.
- 다시 반복조건인 cond 를 검사하여 반복한다.

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

7.3 분기문

1 부터 15 까지 5 의 배구가 아닌 정수 출력

```c
#include <stdio.h>
int main(void) {
        const int MAX = 15;
        printf("1 에서 %d 까지 정수 중에서 5 로 나누어 떨어지지 않는 수\n", MAX);
        for (int i = 1; i < MAX; i++) {
                if (i % 5 == 0)
                        continue;
                printf("%3d", i);
        }
        puts("");
        return 0;
}
```

결과

```
1 에서 15 까지 정수 중에서 5 로 나누어 떨어지지 않는 수
  1  2  3  4  6  7  8  9 11 12 13 14
```

분기문은 정해진 부분으로 바로 실행을 이동하는 기능을 수행한다. C 가 지원하는 분기문으로는 break, continue, goto, return 문이 있다. 특정 구문의 실행을 종료하는 break 는 반복이나 switch 문에서 사용되며, continue 문은 반복에서 사용되어 다음 반복으로 이동하여 실행된다. 특정 레이블이 있는 위치로 실행을 이동하는 goto 문은 어디에서나 사용이 가능하다.

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

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.

goto 문은 레이블이 위치한 다음 문장으로 실행순서를 이동하는 문장이다. 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. 개발자가 의도하지 않은 무한반복은 프로그램이 종료되지 않는 결과가 발생한다. while 과 do while 은 반복조건이 아예 없으면 오류가 발생하니 주의하도록 하자.

7.4 중첩된 반복문

구구단 출력

```c
#include <stdio.h>
#define MAX 9
int main(void) {
	printf("=== 구구단 출력 ===\n");
	for (int i = 2; i <= MAX; i++) {
		printf("%6d 단 출력\n", i);
		for (int j = 2; j <= MAX; j++) {
			printf("%d*%d = %2d", i, j, i*j);
		}
		printf("\n");
	}
	return 0;
}
```

결과

```
=== 구구단 출력 ===
   2 단 출력
2*2 =  4 2*3 =  6 2*4 =  8 2*5 = 10 2*6 = 12 2*7 = 14 2*8 = 16 2*9 = 18
   3 단 출력
3*2 =  6 3*3 =  9 3*4 = 12 3*5 = 15 3*6 = 18 3*7 = 21 3*8 = 24 3*9 = 27
   4 단 출력
4*2 =  8 4*3 = 12 4*4 = 16 4*5 = 20 4*6 = 24 4*7 = 28 4*8 = 32 4*9 = 36
   5 단 출력
5*2 = 10 5*3 = 15 5*4 = 20 5*5 = 25 5*6 = 30 5*7 = 35 5*8 = 40 5*9 = 45
   6 단 출력
6*2 = 12 6*3 = 18 6*4 = 24 6*5 = 30 6*6 = 36 6*7 = 42 6*8 = 48 6*9 = 54
   7 단 출력
7*2 = 14 7*3 = 21 7*4 = 28 7*5 = 35 7*6 = 42 7*7 = 49 7*8 = 56 7*9 = 63
   8 단 출력
8*2 = 16 8*3 = 24 8*4 = 32 8*5 = 40 8*6 = 48 8*7 = 56 8*8 = 64 8*9 = 72
   9 단 출력
9*2 = 18 9*3 = 27 9*4 = 36 9*5 = 45 9*6 = 54 9*7 = 63 9*8 = 72 9*9 = 81
```

반복문 내부에 반복문이 또 있는 구문을 중첩된 반복문이라 한다.

08 포인터 기초

8.1 포인터 변수와 선언

다양한 자료형 포인터 변수 선언에 의한 주소값 출력

```c
#include <stdio.h>
int main(void)
{
	char c = "@";
	char* pc = &c;
	int m = 10;
	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);
}
```

결과

```
변수명     주소값     저장값
----------------------------
  c    00CFF837         P
  m    00CFF81C        10
  x    00CFF800  5.830000
```

메모리 공간은 8 비트인 1 바이트마다 고유한 주소가 있다. 메모리 주소는 0 부터 바이트마다 1 씩 증가한다. 메모리 주소는 저장 장소인 변수이름과 함께 기억 장소를 참조하는 또 다른 방법이다. &은 피연산자인 변수의 메모리 주소를 반환하는 주소연산자이다.

변수의 주소도 포인터 변수에 저장할 수 있다. 변수의 주소값은 반드시 포인터 변수에 저장해야 한다. 즉 포인터 변수는 주소값을 저장하는 변수로 일반 변수와 구별되며 선언방법이 다르다.

포인터 변수 선언에서 자료형과 포인터 변수 이름 사이에 연산자 *(asterisk)를 삽입한다. 즉 int 형 변수의 주소를 저장하는 포인터는 int*이고, double 형 변수의 주소를 저장하는 포인터는 double*이어야 한다. 즉 어느 변수의 주소값을 저장하려면 반드시 그 변수의 자료유형과 동일한 포인터 변수에 저장해야 한다. 포인터 변수 선언에서 포인터를 의미하는 *는 자료형과 변수 이름 사이에만 위치하면 된다. 일반적으로 변수 이름 앞에 *를 붙이는 방식을 가장 선호한다.

8.2 간접 연산자 *와 포인터 연산

포인터를 이용하여 두 수의 값을 교환하는 프로그램

```
#include <stdio.h>
int main(void)
{
          int m = 100, n = 200, dummy;
          printf("%d %d\n", m, n);
          int* p = &m;
          dummy = *p;
          *p = n;
          p = &n;
          *p = dummy;
          printf("%d %d\n", m, n);
          return 0;
}
```

결과

```
100 200
200 100
```

포인터 변수가 갖는 주소로 그 주소의 원래 변수를 참조할 수 있다. 포인터 변수가 가리키고 있는 변수를 참조하려면 간접연산자 *를 사용한다.

포인터 변수는 간단한 더하기와 뺄셈 연산으로 이웃한 변수의 주소 연산을 수행할 수 있다. 포인터의 연산은 절대적인 주소의 계산이 아니며, 포인터가 가리키는 변수 크기에 비례한 연산이다. 즉 포인터에 저장된 주소값의 연산으로 이웃한 이전 또는 이후의 다른 변수를 참조할 수 있다.

int 형 포인터 pi 에 저장된 주소값이 100 이라 가정하자. 그렇다면 (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. 또한 (pi-1)은 4 만큼 감소한 96 이 된다. 마찬가지로 double 형 포인터 pd 의 값이 100 이라면, (pd+1)은 108 이며, pd-1 은 92 가 된다. 더하기와 빼기 연산에는 포인터 변수가 피연산자로 참여할 수 있다. 그러나 곱하기와 나누기에는 포인터 변수가 피연산자로 참여할 수 있다. 즉 포인터 변수의 나누기와 곱하기 연산은 문법 오류가 발생한다.

8.3 포인터 형변환과 다중 포인터

표준입력으로 받은 두 실수의 덧셈을 포인터 변수를 사용해 수행하고 출력

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int main(void) {
	double x, y;
	double* px = &x;
	double* py = &y;
	printf("두 실수 입력: ");
	scanf("%lf %lf", px, py);
	printf("%.2f + %.2f = %.2f\n", *px, *py, *px + *py);
	return 0;
}
```

결과

```
두 실수 입력: 3.87 7.983
3.87 + 7.98 = 11.85
```

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

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

키워드 const 를 이용하여 변수 선언은 변수를 상수로 만들 듯이 포인터 변수도 포인터 상수로 만들 수 있다. 포인터 변수 선언에서 키워드 const 를 삽입하는 방법은 다음과 같이 3 종류가 있을 수 있으나 첫 번째와 두 번째는 같은 의미이므로 두 가지 방식이 있다고 볼 수 있다.

```c
int  i = 10, j = 20;
const int* pi = &i;
int const* pi = &i;
int* const pi = &i;
```

09 배열

9.1 배열 선언과 초기화

정수 int 형 배열에 표준입력으로 받은 정수를 저장하여 출력

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int main(void) {
	int input[20] = { 0 };
	printf("배열에 저장할 정수를 여러 개 입력하시오.");
	printf(" 0 을 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++]);
	}
	return 0;
}
```

결과

```
배열에 저장할 정수를 여러 개 입력하시오. 0 을 입력하면 입력을 종료합니다.
30 26 65 39 87 76 0
30 26 65 39 87 76
```

배열은 동일한 자료 유형이 여러 개 필요한 경우에 유용한 자료 구조이다. 즉 배열은 한 자료유형의 저장공간인 원소를 동일하 크기로 지정된 배열 크기만큼 확보한 연속된 저장공간이다. 배열은 변수이므로 배열마다 고유한 배열이름을 갖는다. 배열을 구성하는 각각의 항목을 배열의 원소라 한다. 그러므로 배열에서 중요한 요소는 배열이름, 원소 자료유형, 배열 크기이다. 배열원소는 첨자 번호라는 숫자를 이용해 쉽게 접근할 수 있다. 이것은 아파트를 호수로 쉽게 찾을 수 있는것과 같은 개념이라고 볼 수 있다.

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

9.2 이차원과 삼차원 배열

이차원 배열에서 원소참조, 주소와 저장값을 출력

```c
#include <stdio.h>
int main(void) {
	int a[3][4] = {
		{1,2,7,3},
		{5,6,3,5},
		{9,7,1,8}
	};
	printf("%6s %6s %3s ", "원소", "주소", "값");
	printf("%6s %6s %3s ", "원소", "주소", "값");
	printf("%6s %6s %3s ", "원소", "주소", "값");
	printf("%6s %6s %3s\n", "원소", "주소", "값");
	for (int i = 0; i < 3; i++) {
		for (int j = 0; j < 4; j++)
			printf("a[%d][%d] %d %d ", i, j, &a[i][j], a[i][j]);
		puts("");
	}
}
```

결과

```
원소  주소 값  원소  주소 값  원소  주소 값  원소  주소 값
a[0][0] 7600676 1 a[0][1] 7600680 2 a[0][2] 7600684 7 a[0][3] 7600688 3
a[1][0] 7600692 5 a[1][1] 7600696 6 a[1][2] 7600700 3 a[1][3] 7600704 5
a[2][0] 7600708 9 a[2][1] 7600712 7 a[2][2] 7600716 1 a[2][3] 7600720 8
```

이차원 배열에서 각 원소를 참조하기 위해서는 2 개의 첨자가 필요하다. 배열선언 int td[2][3];으로 선언된 배열 td 에서 첫 번째 원소는 td[0][0]로 참조 된다. 이어서 두 번째 원소는 td[0][1]로 열의 첨자가 1 증가한다.

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.

이차원 배열을 선언하면서 초기값을 지정하는 방법은 중괄호를 중첩되게 이용하는 방법과 일차원 배열 같이 하나의 중괄호를 사용하는 방법이 있다. 중괄호 내부에 행에 속하는 값을 다시 중괄호로 묶고, 중괄호와 중괄호 사이에는 쉼표로 분리한다. 행인 중괄호 내부의 초기값들은 쉼표로 분리한다. 이 방법은 이차원 구조를 행과 열로 표현할 수 있는 장점이 있다.

이차원 배열선언 초기값 지정의 다른 방법으로는 일차원 배열과 같이 하나의 중괄호로 모든 초기값을 쉼표로 분리하는 방법이 있다.

9.3 배열과 포인터 관계

실수를 위한 배열 크기가 2 인 double 형 배열 내부에 int 형 자료값을 4 개 저장하여 이 정수가 저장된 공간의 주소와 저장값을 출력

```
#include <stdio.h>
int main(void) {
          double dint[4] = { 0.0 };
          int* p = (int*)dint;
          p[0] = 1;
          p[1] = 2;
          p[2] = 3;
          p[3] = 4;
          for(int i = 0; i < 4; i++)
                    printf("%p %d\n", p + i, *(p + i));
          return 0;
}
```

결과

```
001AFEB4 1
001AFEB8 2
001AFEBC 3
001AFEC0 4
```

배열은 실제 포인터와 연관성이 매우 많다. 다음 배열 score 에서 배열이름 score 자체가 배열 첫 원소의 주소값인 상수이다. 다음과 같은 특징으로 배열이름 score 를 이용하여 모든 배열원소의 주소와 저장값을 참조할 수 있다.

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

- 배열 이름 score 는 배열 첫번째의 원소를 주소를 나타내는 상수로 &score[0]와 같으며 배열을 대표한다. 그러므로 간접연산자를 이용한 *score 는 변수 score[0]와 같다.
- 배열 이름 score 가 포인터 상수로 연산식 (score + 1)이 가능하다. 즉 연산식 (score+1)은 score 의 다음 배열 원소의 주소값을 의미한다. 즉 (score+1)은 &score[1]이다. 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.

9.4 포인터 배열과 배열 포인터

이차원 배열에서 각 행의 첫 주소와 2 행의 모든 원소의 주소와 값을 출력

```
#include <stdio.h>
int main(void) {
	int abc[4][3] = {
		{ 1, 2, 3 },
		{ 5, 6, 7 },
		{ 9, 10, 11 },
		{ 13, 14, 15 },
	};
	int rowsize = sizeof(abc) / sizeof(abc[0]);
	int colsize = sizeof(abc[0]) / sizeof(abc[0][0]);
	printf("각 행의 첫 주소 출력: \n");
	for (int i = 0; i < rowsize; i++) {
		printf("%p ", abc[i]);
	}
	printf("\n\n");
	printf("2 행 원소의 주소와 값 출력: \n");
	int* p = abc[1];
	for (int i = 0; i < colsize; i++) {
		printf("%p ", p);
		printf("%d\n", *p++);
	}
}
```

결과

```
각 행의 첫 주소 출력:
00F2F914 00F2F920 00F2F92C 00F2F938

2 행 원소의 주소와 값 출력:
00F2F920 5
00F2F924 6
00F2F928 7
```

일반 변수의 배열이 있듯이 포인터 배열이란 주소값을 저장하는 포인터를 배열 원소로 하는 배열이다. 일반 선언에서 변수이름 앞에 *를 붙이면 포인터 배열 변수 선언이 된다.

열이 4 인 이차원 배열 arr[][4]의 주소를 저장하려면 배열 포인터 변수 ptr 을 문장 int (*ptr)[4]; 로 선언해야 한다. 여기서 대괄호 사이의 4 는 가리키는 이차원 배열에서의 열 크기이다. 즉 이차원 배열의 주소를 저장하는 포인터 변수는 열 크기에 따라 변수 선언이 달라진다.

10 함수기초

10.1 함수정의와 호출

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

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int main(void) {
	int max = 0;
	printf("1 에서 n 까지의 합을 구할 n 을 입력하시오. >> ");
	scanf("%d", &max);
	printf("1 에서 %d 까지의 합 : %d\n", max , getsum(max));
	return 0;
}
int getsum(int n) {
	int sum = 0;
	for (int i = 1; i <= n; i++) {
		sum += i;
	}
	return sum;
}
```

결과

```
각 행의 첫 주소 출력:
00F2F914 00F2F920 00F2F92C 00F2F938

2 행 원소의 주소와 값 출력:
00F2F920 5
00F2F924 6
00F2F928 7
```

- 함수머리는 반환형과 함수이름, 매개변수 목록으로 구성된다.
- 함수머리에서 반환형은 함수 결과값이 자료형으로 간단히 반환형이라 부른다. 이 반환형은 int, float, double 과 같은 다양한 자료형이 올 수 있다.
- 함수이름은 식별자의 생성 규칙을 따른다. 괄호 안에 기술되는 매개변수 목록은 자료형 변수이름의 쌍으로 필요한 수만큼 콤마로 구분하여 기술된다.
- 함수몸체는 {...}와 같이 중괄호로 시작하여 중괄호로 종료된다.
- 함수몸체에서는 함수가 수행해야할 문장들로 구성된다.

10.2 함수의 매개변수 활용

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

```c
#include <stdio.h>
void incrementary(int ary[], int n, int SIZE);
void printary(int* data, int SIZE);
int main(void) {
	int data[] = { 4,7,2,3,5 };
	int aryLength = sizeof(data) / sizeof(int);
	printary(data, aryLength);
	incrementary(data, 3, aryLength);
	printf("배열 원소에 각각 3 을 더한 결과: \n");
	printary(data, aryLength);
	return 0;
}
void incrementary(int ary[], int n, int SIZE)
{
	for (int i = 0; i < SIZE; i++) {
		*(ary + i) += n;
	}
}
void printary(int* data, int SIZE) {
	for (int i = 0; i < SIZE; i++) {
		printf("%2d ", data[i]);
	}
	printf("\n");
}
```

결과

```
4  7  2  3  5
배열 원소에 각각 3 을 더한 결과:
 7 10  5  6  8
```

함수정의에서 기술되는 매개변수 목록의 변수를 형식매개변수라 한다. 함수의 매개변수로 배열을 전달한다면 한 번에 여러 개의 변수를 전달하는 효과를 가져온다.

다차원 배열을 인자로 이용하는 경우, 함수원형과 함수정의의 헤더에서 첫 번째 대괄호 내부의 크기를 제외한 다른 모든 크기는 반드시 기술되어야 한다. 그러므로 이차원 배열의 행의 수를 인자로 이용하면 보다 일반화된 함수로 구현할 수 있다.

이차원 배열의 행의 수는 다음과 같이 (sizeof(x)/sizeof(x[0])로 계산할 수 있다. 또한 이차원 배열의 열의 수는 다음과 같이 (sizeof(x[0])/sizeof(x[0][0])로 계산할 수 있다. 여기서 sizeof(x)는 배열 전체의 바이트 수를 나타내며 (sizeof(x[0])은 1 행의 바이트 수, sizeof(x[0][0])은 첫 번째 원소의 바이트 수를 나타낸다.

10.3 재귀와 라이브러리 함수

1 에서 100 사이의 난수 알아 맞추기

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 100
int main(void) {
	int guess, input;
	srand((long)time(NULL));
	guess = rand() % MAX + 1;
	printf("1 에서 %d 사이에서 한 정수가 결정되었습니다.\n", MAX);
	printf("이 정수는 무엇일까요? 입력해 주세요: ");
	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;
}
```

결과

```
1 에서 100 사이에서 한 정수가 결정되었습니다.
이 정수는 무엇일까요? 입력해 주세요: 50
입력한 수보다 작습니다. 다시 입력하세요 : 25
입력한 수보다 작습니다. 다시 입력하세요 : 12
입력한 수보다 큽니다. 다시 입력하세요 : 18
입력한 수보다 큽니다. 다시 입력하세요 : 22
정답입니다.
```

함수 rand()는 함수호출 순서에 따라 항상 일정한 수가 반환된다. 매번 다르게 생성하려면 시드값을 이용해야 한다.

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

11 문자와 문자열

11.1 문자와 문자열

한 행을 표준입력으로 받아 문자 하나 하나를 그대로 출력

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int main(void) {
          char s[100];
          gets(s);
          char* p = s;
          while (*p) {
                    printf("%c", *p++);
          }
          printf("\n");
          return 0;
}
```

결과

```
int main(void)
int main(void)
```

함수 getchar()는 문자의 입력에 사용되고 putchar()는 문자의 출력에 사용된다.

함수 getche()는 버퍼를 사용하지 않으므로 문자 하나를 입력하면 바로 함수 getche()가 실행된다. 함수 getche()에서 입력된 문자는 바로 모니터에 나타난다.

getch()는 문자입력을 위한 것으로, 입력한 문자가 화면에 보이지 않는 특성이 있다.

함수 scanf()는 공백으로 구분되는 하나의 문자열을 입력 받을 수 있다.

- 가장 먼저 입력 받은 문자열이 저장될 충분한 공간인 문자 배열 str 을 선언한다.
- 함수 scnaf(%s", str)에서 형식제어문자 %s 를 사용하여 문자열을 입력 받을 수 있다.
- 함수 printf("%s", str)에서 %s 를 사용하여 문자열을 출력한다.

함수 gets()는 한 행의 문자열 입력에 유용한 함수이다. 또한 puts()는 한 행에 문자열을 출력하는 함수이다.

11.2 문자열 관련 함수

문자열을 역순으로 저장하는 함수 reverse() 구현

```c
#include <stdio.h>
#include <string.h>
void reverse(char str[]);
int main(void) {
	char s[50];
	memcpy(s, "C Programming!", strlen("C Programming!") + 1);
	reverse(s);
	printf("%s\n", s);
	return 0;
}
void reverse(char str[]) {
	for (int i = 0, j = strlen(str) - 1; i < j; i++, j--) {
		char c = str[i];
		str[i] = str[j];
		str[j] = c;
	}
}
```

결과

```
!gnimmargorP C
```

문자열 비교와 복사, 그리고 문자열 연결 등과 같은 다양한 문자열 처리는 헤더파일 string.h 에 함수원형으로 선언된 라이브러리 함수로 제공된다. 함수에서 사용되는 자료형 size_t 는 비부호 정수형이며, void*는 아직 정해지지 않은 다양한 포인터를 의미한다.

함수 strcmp()는 인자인 두 문자열을 사전 상의 순서로 비교하는 함수이다.

함수 strcpy()와 strncpy()는 문자열을 복사하는 함수이다. 함수 strcpy()는 앞 인자 문자열 dest 에 뒤 인자 문자열 source 를 복사한다.

함수 strcat()는 하나의 문자열 되에 다른 하나의 문자열을 연이어 추가해 연결할 때 쓰인다.

함수 strtok()는 문자열에서 구분자인 문자를 여러 개 지정하여 토큰을 추출하는 함수이다.

함수 strlen()은 NULL 문자를 제외한 문자열 길이를 반환하는 함수이다.

11.3 여러 문자열 처리

여러 문자열 처리

```c
#include <stdio.h>
int main(void) {
        char str1[] = "JAVA";
        char str2[] = "C#";
        char str3[] = "C++";
        char* pstr[] = { str1,str2,str3 };
        // 각각의 3 개 문자열 출력
        printf("%s ", pstr[0]);
        printf("%s ", pstr[1]);
        printf("%s\n", pstr[2]);
        // 문자 출력
        printf("%c %c %c\n", str1[0], str2[1], str3[2]);
        printf("%c %c %c\n", pstr[0][1], pstr[1][1], pstr[2][1]);
}
```

결과

```
JAVA C# C++
J # +
A # +
```

여러 개의 문자열을 처리하는 하나의 방법은 문자 포인터 배열을 이용하는 방법이다. 하나의 문자 포인터가 하나의 문자열을 참조할 수 있으므로 문자 포인터 배열은 여러 개의 문자열을 참조할 수 있다.

여러 개의 문자열을 처리하는 다른 방법은 문자의 이차원 배열을 이용하는 방법이다.
배열선언에서 이차원 배열의 열 크기는 문자열 중에서 가장 긴 문자열의 길이보다 1 크게 지정해야 한다.

프로그램에서 명령행 인자는 main()함수의 인자로 기술된다.

- 프로그램에서 명령행 인자를 받으려면 main() 함수에서 두 개의 인자 argc 와 argv 를 (int argc, char * argv[])로 기술해야 한다.
- 매개변수 argc 는 명령행에서 입력한 문자열의 수이며 argv[]는 명령행에서 입력한 문자열을 전달 받는 문자 포인터 배열이다.
- 여기서 주의할 점은 실행 프로그램 이름도 하나의 명령행 인자에 포함된다는 사실이다.

12 변수 유효범위

12.1 전역변수와 지역변수

피보나츠 수의 출력

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int count;
void fibonacci(int prev_number, int number);
int main(void) {
	auto prev_number = 0, number = 1;
	printf("피보나츠를 몇 개 구할까요(3 이상) >> ");
	scanf("%d", &count);
	if (count <= 2)
		return 0;
	printf("1 ");
	fibonacci(prev_number, number);
	printf("\n");
	return 0;
}
void fibonacci(int prev_number, int number) {
	static int i = 1;

	while (i++ < count) {
		int next_num = prev_number + number;
		prev_number = number;
		number = next_num;
		printf("%d ", next_num);
		fibonacci(prev_number, number);
	}
}
```

결과

```
피보나츠를 몇 개 구할까요(3 이상) >> 20
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
```

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

지역변수는 함수 또는 블록에서 선언된 변수이다. 지역변수는 내부변수, 자동변수라고도 한다.

전역변수는 함수 위부에서 선언된 변수이다. 외부변수라고도 부른다. 일반적으로 프로젝트의 모든 함수나 블록에서 참조할 수 있다.

12.2 정적 변수와 레지스터 변수

지역변수와 정적변수의 사용

```c
#include <stdio.h>
void process();
int main(void)
{
	process();
	process();
	process();
	return 0;
}
void process()
{
	// 정적변수
	static int sx;
	// 지역 변수
	int x = 1;
	printf("%d %d\n", x, sx);
	x += 3;
	sx += x + 3;
}
```

결과

```
1 0
1 7
1 14
```

register 변수는 변수의 저장공간이 일반 메모리가 아니라 CPU 내부의 레지스터에 할당되는 변수이다.

static 변수는 정적변수이다. 정적변수는 초기 생성된 이후 메모리에서 제거되지 않으므로 지속적으로 저장값을 유지하거나 수정할 수 있는 특성이 있다.

함수나 블록에서 정적으로 선언되는 변수가 정적 지역변수이다. 정적 지역분수의 유효 범위는 선언된 블록 내부에서만 참조 가능하다. 정적 지역변수는 함수나 블록을 종료해도 메모리에서 제거되지 않고 계속 메모리에 유지 관리되는 특성이 있다.

함수 외부에서 정적으로 선언되는 변수가 정적 전역변수이다. 일반 전역변수는 파일 소스가 다르더라도 extern 을 사용하여 참조가 가능하다. 그러나 정적 전역변수는 선언된 파일 내부에서만 참조가 가능한 변수이다. 즉 정적 전역변수는 extern 에 의해 다른 파일에서 참조가 불가능하다.

12.3 메모리 영역과 변수이용

은행계좌의 입출금 구현

```c
#include <stdio.h>
// 전역변수
int total = 10000;
// 입금 함수원형
void save(int);
//출금 함수원형
void withdraw(int);
int main(void) {
	printf(" 입금액  출금액   총입금액   총출금액    잔고\n");
	printf("=========================================================\n");
	printf("%46d\n", total);
	save(50000);
	withdraw(30000);
	save(60000);
	withdraw(20000);
	printf("=========================================================\n");
	return 0;
}
// 입금액을 매개변수로 사용
void save(int money) {
	// 총입금액이 저장되는 정적 지역변수
	static int amount;
	total += money;
	amount += money;
	printf("%7d %17d %20d\n", money, amount, total);
}
// 출금액을 매개변수로 사용
void withdraw(int money) {
	// 총출금액이 저장되는 정적 지역변수
	static int amount;
	total -= money;
	amount += money;
	printf("%15d %20d %9d\n", money, amount, total);
}
```

결과

```
입금액   출금액    총입금액    총출금액     잔고
=========================================================
                                        10000
  50000              50000              60000
          30000                30000     30000
  60000             110000              90000
          20000                50000     70000
=========================================================
```

13 구조체와 공용체

13.1 구조체와 공용체

도시의 이름과 위치를 표현하는 구조체

```
#include <stdio.h>
#include <string.h>
// 지구 위치 구조체
struct position
{
	// 위도
	double latitude;
	// 경도
	double longitude;
};
int main(void)
{
	// 도시 정보 구조체
	struct city
	{
		// 이름
		char* name;
		// 위치
		struct position place;
	};
	struct city seoul, newyork;
	seoul.name = "서울";
	seoul.place.latitude = 37.33;
	seoul.place.longitude = 126.58;
	newyork.name = "뉴욕";
	newyork.place.latitude = 40.8;
	newyork.place.longitude = 73.9;
	printf("[%s] 위도=%.1f 경도= %.1f\n", seoul.name, seoul.place.longitude,
seoul.place.latitude);
	printf("[%s] 위도=%.1f 경도= %.1f\n", newyork.name, newyork.place.longitude,
newyork.place.latitude);
	return 0;
}
```

결과

```
[서울] 위도=126.6 경도= 37.3
[뉴욕] 위도=73.9 경도= 40.8
```

13.2 자료형 재정의

도시의 이름과 위치를 표현하는 구조체

```c
#include <stdio.h>
int main(void)
{
	// 영화 정보 구조체
	typedef struct movie
	{
		// 영화제목
		char* title;
		// 관객수
		int attendance;
	} movie;
	movie assassination;
	assassination.title = "암살";
	assassination.attendance = 12700000;
	printf("[%s] 관객수: %d\n", assassination.title, assassination.attendance);
	return 0;
}
```

결과

```
[암살] 관객수: 12700000
```

typedef 는 이미 사용되는 자료 유형을 다른 새로운 자료형 이름으로 재정의할 수 있도록 하는 키워드이다. 문장 typedef int profit;은 profit 을 int 와 같은 자료형을 새롭게 정의하는 문장이다.

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

문장 typedef 도 일반 변수와 같이 그 사용 범위를 제한한다. 그러므로 함수 내부에서 재정의 되면 선언된 이후의 그 함수에서만 이용이 가능하다. 만일 함수 위부에서 재정의된다면 재정의된 이후 그 파일에서 이용이 가능하다.

구조체 struct data 가 정의된 상태에서 typedef 사용하여 구조체 struct data 를 data 로 재정의할 수 있다. 이제 새로운 자료유형인 date 는 struct date 와 함께 동일한 자료유형으로 이용이 가능하다. 물론 date 가 아닌 datetype 등 다른 이름으로도 재정의 가능하다.

typedef 를 이용하는 다른 방법은 구조체 정의 자체를 typedef 와 함께 처리하는 방법이다.

13.3 구조체와 공용체의 포인터와 배열

영화 정보를 표현하는 구조체와 배열

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>
int main(void)
{
	// 영화 정보 구조체
	typedef struct movie
	{
		// 영화제목
		char* title;
		// 관객수
		int attendance;
		// 감독
		char director[20];
	} movie;
	movie box[] =
	{
		{ "명량", 17613000, "김한민" },
		{ "국제시장", 14257000, "윤제균" },
		{ "베테랑", 13383000 }
	};
	strcpy(box[2].director, "류승완");
	printf("    제목    감독    관객수\n");
	printf("=============================================\n");
	for (int i = 0; i < 3; i++)
	{
		printf("[%8s] %6s %d\n", box[i].title, box[i].director,
box[i].attendance);
	}
	return 0;
}
```

결과

```
제목     감독    관객수
=============================================
[    명량] 김한민 17613000
[국제시장] 윤제균 14257000
[  베테랑] 류승완 13383000
```

구조체 포인터 멤버 접근 연산자 ->는 p->name 과 같이 사용한다. 연산식 p->name 은 포인터 p 가 가리키는 구조체 변수의 멤버 name 을 접근하는 연산식이다.

연산식 *p.name 은 접근연산자(.)가 간접연산자(*)보다 우선순위가 빠르므로 *(p.name)과 같은 연산식이다. p 가 포인터이므로 p.name 는 문법 오류가 발생한다.