

C 어플리케이션구현 포트폴리오

컴퓨터정보공학과

홍나리

20193422

01. CHAPTER

프로그래밍 개요

프로그램이 무엇일까

컴퓨터와 스마트폰에서 특정 목적의 작업을 수행하기 위한 관련 파일의 모임을 프로그램이라고 한다.

프로그래머와 프로그래밍 언어

컴퓨터와 스마트폰 등의 정보기기에서 사용되는 프로그램을 만드는 사람을 프로그래머(Programmer)라고 한다.

프로그래머가 프로그램을 개발하기 위해 사용하는 언어가 프로그래밍 언어이다.

사람과 컴퓨터가 서로 의사교환을 하기위한 언어를 프로그래밍 언어라고 하며

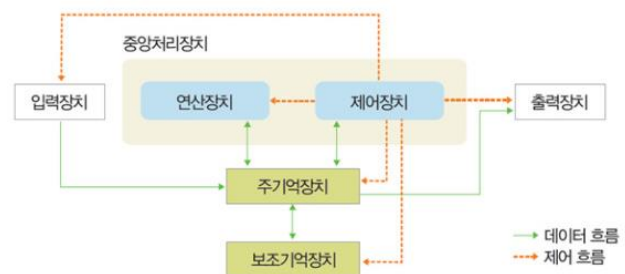
사람이 컴퓨터에게 지시할 명령어를 기술하기 위하여 만들어진 언어이다.

포트란, 베이직, 코볼, 파스칼, C, C++, Java, JSP, 파이썬 등 매우 다양하다.

하드웨어와 소프트웨어

컴퓨터는 물리적인 하드웨어와 프로그램인 소프트웨어로 이루어져 있다.

하드웨어의 중요한 구성 요소로는 중앙처리장치(CPU), 주기억장치, 보조기억장치, 입력장치, 출력장치가 있다.



소프트웨어는 컴퓨터가 수행할 작업을

지시하는 전자적 명령어들의 집합으로 구성된 프로그램을 말하며, 컴퓨터의 하드웨어가 해야 할 작업내용을 지시한다.

소프트웨어는 크게 응용소프트웨어와 시스템소프트웨어로 나뉜다.

응용소프트웨어는 문서 작성이나 인터넷검색, 게임 등 특정 업무에 활용되는 소프트웨어를 말한다.

시스템 소프트웨어는 운영체제와 각종 유틸리티 프로그램으로 구분되는데,

운영체제는 특정 CPU에 맞게 관련된 하드웨어를 작동하게 하고 또한 응용소프트웨어를 실행해주는 소프트웨어이다.

저급언어와 고급언어

컴퓨터의 CPU 에 적합하게 만든 기계어와 어셈블리 언어를 모두 **저급언어**라고 한다.

기계어는 컴퓨터가 직접 이해할 수 있는 유일한 언어로 전기의 흐름을 표현하는 1 과 흐르지 않음을 의미하는 0 으로 표현된다.

어셈블리어는 기계어를 프로그래머인 사람이 좀 더 이해하기 쉬운 기호 형태로 1:1 대응시킨 프로그래밍 언어이다.

저급언어와 반대로 **고급언어**는 컴퓨터의 CPU 에 의존하지 않고 사람이 보다 쉽게 이해할 수 있도록 만들어진 언어이다.

고급언어인 프로그래밍 언어로는 C 언어를 비롯하여 포트란, 파스칼, 베이직, 자바 등 다양하다.

이런 고급언어들은 반드시 기계어로 변환되어야 실행 가능하다.

고급언어로 작성된 프로그램을 기계어 또는 목적코드로 바꾸어 주는 프로그램이 바로 **컴파일러**이다.

마찬가지로 어셈블리어로 작성된 프로그램을 기계어로 바꾸어 주는 프로그램은 어셈블러이다.

C 언어란

C 언어는 1972 년 데니스 리치가 개발한 프로그래밍 언어이다.

유닉스를 개발하기 위해 C 언어를 개발하였는데

어셈블리 언어 정도의 속도를 내며, 좀 더 쉽고, 서로 다른 CPU 에서도 작동되는 프로그래밍 언어가 필요해서 만든 언어이다.

C 언어는 1970 년에 개발한 B 언어에서 유래된 프로그래밍 언어이다.

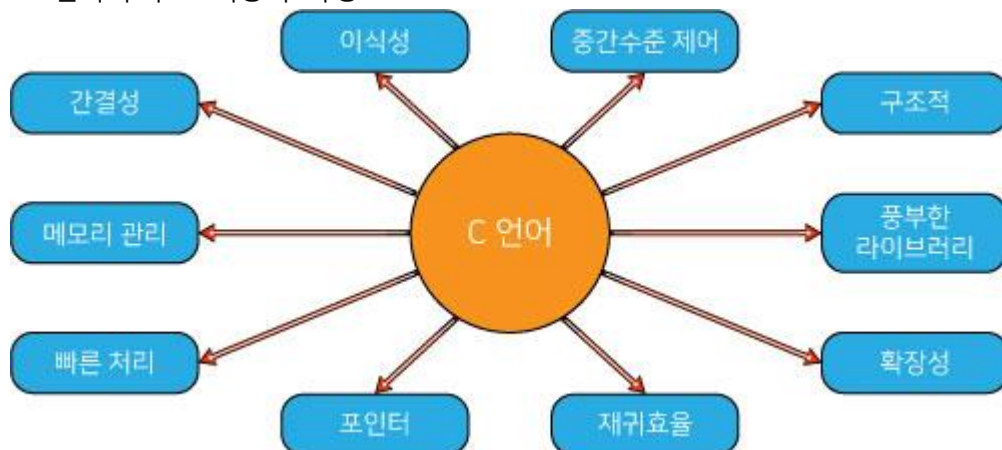
C 언어의 큰 특징 중 하나는 C 언어는 **절차지향언어** 라는 점이다.

절차지향 언어란 시간의 흐름에 따라 정해진 절차를 실행한다는 의미로

C 언어는 문제의 해결 순서와 절차의 표현과 해결이 쉽도록 설계된 프로그램 언어이다.

또 C 언어는 **간결하고 효율적인** 언어이다. 그 이유는 아래 그림을 보면 알 수 있다.

<C 언어의 주요 기능과 특징>



C 언어는 **이식성**이 좋다.

C 로 작성된 소스는 별 다른 수정없이 다양한 운영체제의 여러 플랫폼에서 제공되는 컴파일러로 번역해 실행 될 수 있다.

C 언어는 다양한 CPU 와 플랫폼의 컴파일러를 지원하기 때문에 이식성이 좋은 것이다.

하지만 C 언어는 배우기에 다소 어렵다.

C 언어의 문법이 상대적으로 간결하지만 많은 내용을 함축하고있으며,

비트와 포인터의 개념 등의 이유로 조금 어렵지만 다른 프로그래밍 언어에도 영향을 많이 끼친 언어로 한번 익히면 다른 프로그래밍 언어 습득에 많은 도움을 주고있다.

그렇기 때문에 현재에도 가장 먼저 배우는 프로그래밍 언어가 주로 C 언어인 것이다.

프로그래밍의 자료 표현

일반적으로 우리가 사용하는 숫자는 십진수이다.

수에서 하나의 자릿수에 사용하는 숫자가 0~9 까지 총 열개인 수가 십진수이다.

하지만 우리는 십진수 외의 다양한 진수를 흔하게 사용한다.

시계는 12 진수 혹은 24 진수로 사용하고 분은 60 진수, 일주는 7 진수로 사용하고있다.

컴퓨터 시스템 내부에서는 이진수를 사용하여 저장한다.

디지털 신호에서 전기가 흐를 경우 참을 의미하는 1, 흐르지 않을 경우 거짓의 0 으로 표현된다.

이진수에서 가장 오른쪽은 단(2^0)단위이며, 왼쪽으로 갈수록 $2(2^1)$ 단위, $4(2^2)$ 단위, $8(2^3)$ 단위, $16(2^4)$ 단위 등이다.







| 단위 | 2^4 단위 | 2^3 단위 | 2^2 단위 | 2^1 단위 | 2^0 단위 | |
|-------|----------|----------|----------|----------|----------|-----|
| 2 진수 | 1 | 1 | 1 | 1 | 1 | |
| 10 진수 | 16+ | 8+ | 4+ | 2+ | 1 | =31 |

정보 처리 단위 중에서 가장 작은 기본 정보 단위가 **비트**(BIT)이다.

비트가 연속적으로 8 개 모인 정보단위를 **바이트**라고 한다.

참(true)과 거짓(false)을 의미하는 두 가지 정보를 **논리값**이라고 한다.

이진 논리 변수와 AND, OR, NOT 의 논리 연산을 이용한 부울 대수는 컴퓨터가 정보를 처리하는 방식에 대하여 이론적인 배경을 제공하며, 0 과 1 두 값 중 하나로 한정된 변수들의 상관 관계를 AND, OR, NOT 등의 여러 연산자를 이용하여 논리적으로 나타낸다.

| 논리동작 | 기호 | 논리식 | 진리표 | | | | | | | | | | | | | | | |
|------|---|----------------------------|--|---|-------|---------|---|---|---|---|---|---|---|---|---|---|---|---|
| NOT |  | $Z = \overline{A}$ | <table><tr><th>A</th><th>NOT B</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table> | A | NOT B | 0 | 1 | 1 | 0 | | | | | | | | | |
| A | NOT B | | | | | | | | | | | | | | | | | |
| 0 | 1 | | | | | | | | | | | | | | | | | |
| 1 | 0 | | | | | | | | | | | | | | | | | |
| AND |  | $Z = A \cdot B$ | <table><tr><th>A</th><th>B</th><th>A and B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table> | A | B | A and B | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| A | B | A and B | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | | | | | | | | | | | | | | | | |
| 0 | 1 | 0 | | | | | | | | | | | | | | | | |
| 1 | 0 | 0 | | | | | | | | | | | | | | | | |
| 1 | 1 | 1 | | | | | | | | | | | | | | | | |
| OR |  | $Z = A+B$ | <table><tr><th>A</th><th>B</th><th>A or B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table> | A | B | A or B | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| A | B | A or B | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | | | | | | | | | | | | | | | | |
| 0 | 1 | 1 | | | | | | | | | | | | | | | | |
| 1 | 0 | 1 | | | | | | | | | | | | | | | | |
| 1 | 1 | 1 | | | | | | | | | | | | | | | | |
| NAND |  | $Z = \overline{A \cdot B}$ | <table><tr><th>A</th><th>B</th><th>A and B</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table> | A | B | A and B | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| A | B | A and B | | | | | | | | | | | | | | | | |
| 0 | 0 | 1 | | | | | | | | | | | | | | | | |
| 0 | 1 | 1 | | | | | | | | | | | | | | | | |
| 1 | 0 | 1 | | | | | | | | | | | | | | | | |
| 1 | 1 | 0 | | | | | | | | | | | | | | | | |
| NOR |  | $Z = \overline{A+B}$ | <table><tr><th>A</th><th>B</th><th>A or B</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table> | A | B | A or B | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| A | B | A or B | | | | | | | | | | | | | | | | |
| 0 | 0 | 1 | | | | | | | | | | | | | | | | |
| 0 | 1 | 0 | | | | | | | | | | | | | | | | |
| 1 | 0 | 0 | | | | | | | | | | | | | | | | |
| 1 | 1 | 0 | | | | | | | | | | | | | | | | |
| XOR |  | $Z = A \oplus B$ | <table><tr><th>A</th><th>B</th><th>A xor B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table> | A | B | A xor B | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| A | B | A xor B | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | | | | | | | | | | | | | | | | |
| 0 | 1 | 1 | | | | | | | | | | | | | | | | |
| 1 | 0 | 1 | | | | | | | | | | | | | | | | |
| 1 | 1 | 0 | | | | | | | | | | | | | | | | |

<논리연산과 회로 그림>

소프트웨어 개발

소프트웨어 개발단계

1. 요구사항 분석
2. 설계
3. 구현
4. 검증
5. 유지보수

1 단계 요구사항 분석에서는 시스템을 사용할 사용자의 요구사항을 파악하여 분석하는 단계이다.

2 단계 설계 단계에서 프로그래머는 알고리즘을 이용하여 소프트웨어를 설계한다.

3 단계 구현단계는 흐름도 또는 의사코드를 컴퓨터가 이해할 수 있는 자바나 C와 같은 특정한 프로그래밍 언어로 개발하는 단계이다.

4 단계 검증단계에서는 프로그램의 소프트웨어 요구사항에 얼마나 부합하는지, 프로그램이 안정적으로 작동하는지를 검사하는 단계이다.

5 단계 프로그램의 문서화 및 유지보수 단계이다.

2 단계인 설계단계에서의 알고리즘 기술방법은 우리가 사용하는 자연어 또는 흐름도나 의사코드를 사용하여 표현할 수 있다.

흐름도란 알고리즘을 표준화된 기호 및 도형으로 도식화하여 데이터의 흐름과 수행되는 연산들의 순서를 표현하는 방법으로 순서도라고도 한다.

02. CHAPTER

통합개발환경과 C 프로그램의 이해

프로그램 구현과정과 통합개발환경

소프트웨어 개발 5 단계 중 3 단계인 구현의 과정은 **프로그램구상>소스편집>컴파일>링크>실행**의 5 단계를 거친다.

프로그램을 개발하기 위해 가장 먼저 해야 할 일은 프로그램 구상과 소스편집이다.

소스 또는 소스코드는 **선정된 프로그래밍 언어인 C 프로그램 자체로 만든 일련의 명령문**을 의미한다.

소스파일은 프로그래밍 언어로 원하는 일련의 명령어가 저장된 파일을 말하며 일반 텍스트파일로 저장되어야 한다.

C 언어의 소스파일 확장자는 .c JAVA 의 소스파일 확장자는 .java 이다.

소스파일에서 기계어로 작성된 목적파일을 만들어내는 프로그램은 **컴파일러**라고 한다.

컴파일러에 의해 처리되기 전의 프로그램이 소스코드라면 컴파일러에 의해 기계어로 번역된 프로그램은 **목적코드**라고 한다.

링커는 하나 이상의 목적파일을 하나의 실행파일로 만들어주는 프로그램이다.

여러 개의 목적파일을 연결하고 참조하는 라이브러리를 포함시켜 하나의 실행파일을 생성하는데 이 과정을 링커 또는 링킹이라고 한다.

자주 사용하는 프로그램들은 프로그램을 작성할 때, 프로그래머마다 새로 작성할 필요없이 개발환경에서 미리 만들어 컴파일해 저장해 놓는데, 이 모듈을 **라이브러리**라고 한다.

라이브러리란 공용으로 사용하기위해 이미 만든 목적코드로 파일.lib 또는 .dll 등으로 제공된다.

비주얼 스튜디오에서는 **컴파일과 링크과정을 하나로 합쳐 빌드**라고 한다.

프로그램 개발과정에서 나타나는 모든 문제를 오류 또는 에러라고 한다.

오류는 **발생시점에 따라 컴파일(시간)오류와 링크(시간)오류, 실행(시간)오류**로 구분할 수 있다.

컴파일오류와 링크오류 사이는 대부분 프로그래밍 언어의 문법을 잘못 기술하여 발생하는 **문법오류**이다.

실행오류는 일부 함수 사용의 잘못이나 입출력문제 등으로 실행 시 에러가 발생하여 프로그램 실행이 중단된다.

목적파일에서 오류는 주로 에러 메시지를 보고 소스를 수정하고 실행파일에서 오류는 링크 에러 메시지를 참고하여 소스를 수정하고 실행 후 결과에서의 오류는 결과를 참조하여 문법오류나 로직오류를 찾아 소스를 수정한다.

오류의 원인과 성격에 따라, 프로그래밍 언어 문법을 잘못 기술한 **문법오류**와 내부 알고리즘이 잘못되거나 원하는 결과가 나오지 않은 등의 **논리오류**로 분류할 수 있다.

프로그램 개발 과정에서 발생하는 다양한 오류를 찾아 소스를 수정하여 다시 컴파일, 링크, 실행하는 과정을 **디버깅**이라고 하고 이를 도와주는 프로그램을 **디버거**라고 한다.

프로그램 개발에 필요한 편집기, 컴파일러, 링커, 디버거 등을 통합하여 편리하고 효율적으로 제공하는 개발환경을 **통합개발환경**, 영문 약자로 **IDE** 라고 한다.

C 프로그램의 이해와 디버깅 과정

C 프로그램과 같은 절차지향 프로그램은 **함수**로 구성된다.

프로그래머가 직접 만드는 함수를 사용자 정의함수라고 하며 시스템이 미리 만들어 놓은 함수를 **라이브러리 함수**라고 한다.

Main90 은 사용자가 직접 만드는 함수 정의과정이며, puts()는 함수 호출 문장이다.

Puts()는 문자열을 전용으로 출력하는 함수이며 함수 printf("문자열")는 호출 시 전달되는 "문자열"과 같은 다양한 형태의 인자를 적절한 형식으로 출력하는 함수이다.

라이브러리함수 puts()와 printf()를 사용하려면 첫 줄에 #include<stdio.h>를 넣어야한다.

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.

인자인 문자열을 출력하고 다음 줄로 이동하여 출력 위치를 지정하려면 함수 puts() 또는 함수 printf("문자열\n")로 호출해야 한다.

03.CHAPTER

프로그래밍 기초

한 프로젝트는 단 하나의 함수 main()과 다른 여러 함수로 구현되며, 최종적으로 프로젝트이름으로 하나의 실행파일이 만들어진다.

C 프로그램은 적어도 main()함수 하나는 구현되어야 응용프로그램으로 실행될 수 있다.

Main()함수는 프로그램이 실행되면 가장 먼저 시작되는 부분이다.

Main()내부에서는 위에서 아래로, 좌에서 우로, 문장이 위치한 순서대로 실행된다.

키워드란 문법적으로 고유한 의미를 갖는 예약된 단어이다. 프로그램 코드를 작성하는 사람이 이 단어들을 다른 용도로 사용해서는 안된다는 뜻이다.

반대로 **식별자**는 프로그래머가 자기 마음대로 정의해서 사용하는 단어이다.

식별자는 키워드와 비교하여 철자 라든지 대문자 소문자 등 **무엇이라도 달라야 한다**.

식별자 구성 규칙

1. 숫자는 맨 앞에 올 수 없다.
2. 대소문자는 구별된다.
3. 중간에 공백문자가 들어갈 수 없다.
4. 키워드는 식별자로 사용할 수 없다.
5. 알파벳과 _를 제외한 문자는 사용할 수 없다.

프로그래밍 언어에서 컴퓨터에게 명령을 대리는 최소 단위를 문장이라하며, 문장은 마지막에 세미콜론으로 종료된다.

여러 개의 문장을 묶으면 블록이라고 한다.

//는 한줄주석으로 //이후부터 그 줄의 마지막까지 주석으로 인식한다.

블록주석/*...*/은 여러 줄에 걸쳐 설명을 사용할 때 이용한다.

자료형과 변수

C 프로그래밍 언어에서 다루는 다양한 자료도 **기본형, 유도형, 사용자 정의형** 등으로 나눌 수 있으며 기본형은 다시 **정수형, 실수형, 문자형, void**로 나뉘는데 프로그래머는 이러한 자료에 적당한 알고리즘을 적용해 프로그램을 작성한다.

자료형은 프로그래밍 언어에서 자료를 식별하는 종류를 말한다.

프로그래밍에 정수와 실수, 문자등의 자료값을 중간중간에 저장할 공간이 필요하다.

이 저장공간을 변수라고 부르는데, 변수에는 고유한 이름이 붙여지며, 물리적으로 기억장치인 메모리에 위치한다.

변수는 선언된 자료형에 따라 변수의 저장공간 크기와 저장되는 자료 값의 종류가 결정된다.

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

프로그램에서 변수를 사용하려면 원칙적으로 사용 전에 먼저 변수선언 과정이 반드시 필요하다.

변수선언은 자료형을 지정한 후 고유한 이름인 변수이름을 나열하여 표시한다.

변수이름은 관습적으로 소문자를 이용하며, 사용목적에 알맞은 이름으로 특정한 영역에서 중복되지 않게 붙이도록 한다.

하나의 자료형으로 여러 개 변수를 한번에 선언하려면 자료형 이후에 변수이름을 콤마로 나열한다.

변수에 저장 값을 대입할때는 대입연산자 표시인 = 을 사용한다.

대입연산자 = 는 오른쪽에 위치한 값을 이미 선언된 왼쪽 변수에 저장한다 라는 의미이다.

이 대입연산이 있는 문장을 대입문이라 한다.

변수를 선언만 하고 자료값이 아무것도 저장되지 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 요소라고 한다.

문장에서 변수의 의미는 저장공간 자체와 저장공간에 저장된 값으로 나눌 수 있다.

자료형

정수형의 기본 키워드는 int 이다.

int 에서 파생된 자료형이 short 와 long 이다.

Short 는 int 보다 작거나 같고, long 은 int 보다 크거나 같다.

정수형 **int, short, long** 은 양수, 0, 음수를 모두 표현할 수 있다.

그러므로 [부호가 있는]을 의미하는 **signed** 키워드는 정수형 자료형 키워드 앞에 표시될 수 있다. 생략도 가능하다.

Unsigned 는 부호가 없는 정수인 0 과 양수만을 저장할 수 있는 정수 자료형이다.

Unsigned int 에서 int 는 생략 가능하다.

Short 는 2 바이트이며, int 와 long 은 모두 4 바이트이다.

부동소수형을 나타내는 키워드는 **float, double, long double** 세가지이다.

Float 는 4 바이트이며, double 과 long double 은 모두 8 바이트이다.

문자형 자료형은 **char, signed char, unsined char** 세 가지 종류가 있다. 문자형 저장공간 크기는 모두 1 바이트이다.

아스키코드는 ANSI 에서 제정한 정보 교환용 표준 코드로 총 127 개의 문자로 구성된다.

연산자 **sizeof** 를 이용하면 자료형, 변수, 상수의 저장공간 크기를 바이트단위로 알 수 있다.

자료형의 범주에서 벗어난 값을 저장하면 오버플로 또는 언더플로가 발생한다.

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

실수형 float 변수에 1.175E-50 와 같이 부동소수점수가 너무 많아 정밀도가 매우 자세한 수를 저장하면 언더플로가 발생하여 0 이 저장된다.

상수

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

상수는 크게 분리하면 **리터럴 상수**와 **심볼릭 상수**로 구분될 수 있다.

리터럴 상수는 달리 이름이 없이 소스에 그대로 표현해 의미가 전달되는 다양한 자료 값을 말한다.

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.

프로그램에서 리터럴 상수란 소스에 그대로 표현해 의미가 전달되는 다양한 자료값을 말한다.

문자 상수는 문자 하나의 앞 뒤에 작은따옴표를 넣어 표현한다.

함수 printf()에서 문자 상수를 출력하려면 다음과 같이 %c 또는 %C의 형식 제어문자가 포함되는 형식 **제어문자열**을 사용한다.

\\n과 같이 \\(역슬래쉬)와 문자의 조합으로 표현하는 문자를 **이스케이프 문자**라 한다.

정수형 상수는 int, unsigned int, long, unsigned long, long long, unsigned long long 등의 자료형으로 나뉜다.

일반적으로 상수의 정수표현은 십진수로 인식되나, 숫자 0을 정수 앞에 놓으면 팔진수로 인식한다.

0x 또는 0X를 숫자 앞에 높으면 십육진수로 인식한다.

실수는 e 또는 E를 사용하여 10의 지수표현방식으로 나타낼 수 있다. 즉 3.14E+2는 3.14×10^2 을 나타낸다.

실수형 상수도 float, double, long double의 자료형으로 나뉜다. 즉 소수는 double 유형이며, float 상수는 숫자 뒤에 f나 F를 붙인다.

변수선언 시 자료형 또는 변수 앞에 키워드 const가 놓이면 이 변수는 심볼릭 상수가 된다.

상수는 변수선언 시 반드시 초기값을 저장해야 한다.

열거형은 키워드 enum를 사용하여 정수형 상수 목록 집합을 정의하는 자료형이다.

열거형 상수에서 목록 첫 상수의 기본값이 0이며 다음부터 1씩 증가하는 방식으로 상수값이 자동으로 부여된다.

상수 목록에 특정한 정수값을 부분적으로 직접 지정할 수도 있다.

상수값을 지정한 상수는 그 값으로, 따로 지정되지 않은 첫번째 상수는 0이며, 중간 상수는 앞의 상수보다 1씩 증가한 상수값으로 정의된다.

전처리 지시자 #define은 매크로 상수를 정의하는 지시자이다.

#define에 의한 심볼릭 상수도 주로 대문자 이름으로 정의하는데, 이를 매크로 상수라고 부른다.

문자형과 정수형의 최대 최소 상수는 헤더파일 limits에 정의되어 있으며, 부동소수형의 최대 최소 상수는 헤더파일 float.h에 정의되어있다.

04.CHAPTER

전처리와 입출력

전처리

C 언어는 컴파일러가 컴파일 하기 전에 전처리의 전처리과정이 필요하다.

전처리 과정에서 처리되는 문장을 전처리지시자라고 한다.

전처리 지시자는 항상 #으로 시작하고, 마지막에 세미콜린이 없는 등 일반 c 언어 문장과는 구별된다. 전처리 지시자는 꼭 한 줄에 다 적어야한다.

대표적인 헤더파일인 stdio.h 는 printf(), scanf(), putchar(), getchar() 등과 같은 입출력 함수를 위한 함수원형 등이 정의된 헤더파일이다.

헤더파일의 확장자는 h 이다.

전처리 지시자 #define 은 매크로 상수를 정의하는 지시자이다.

#define 에 의한 심볼릭 상수도 주로 대문자 이름으로 정의하는데, 이를 매크로상수라고 부른다.

매크로는 이미 정의된 매크로를 다시 사용할 수 있다.

출력함수

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

Int term = 15

```
Printf("%d 의 두 배는 %d 입니다.\n",term, 2*term);
```

함수 printf()의 첫 번째 인자인 형식문자열은 일반 문자와 이스케이프 문자, 형식 지정자로 구성된다.

정수의 십진수 출력 형식 지정자는 %d 와 %i 이다.

정수를 9 진수로 출력하려면 %o 를 이용하고 앞부분에 0 이 붙는 출력을 하려면 %#o 를 이용한다.

정수를 소문자의 십육진수로 출력하려면 %x 와 대문자로 출력하려면 %X 를 이용하며, 출력되는 16 진수 앞에 0x 또는 0X 를 붙여 출력하려면 #을 삽입하여 %#x 와 %#X 를 이용한다.

함수 printf()는 두번째 인자부터 시작되는 인자의 값을 형식지정자에 맞게 서식화하여 출력하며 반환값은 출력한 문자 수 이며, 오류가 발생하면 음수를 반환한다.

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

입력함수

scanf()는 대표적인 입력함수이고 printf()와 동일한 형식지정자를 사용한다.

&는 주소연산자로 뒤에 표시된 피연산자인 변수 주소값이 연산값으로 scanf()의 입력변수목록에는 키보드에 입력값이 저장되는 변수를 찾는다는 의미에서 반드시 변수의 주소연산식 '&변수이름'이 인자로 사용되어야 한다.

만일 주소연산이 아닌 변수 year 로 기술하면 입력값이 저장될 주소를 찾지못해 오류가 발생한다. 지정된 형식지정자에 맞게 키보드로 적당한 값을 입력한 후 엔터키를 누르기 전까지는 실행을 멈춰 사용자의 입력을 기다린다.

여러 입력값을 구분해주는 구분자로 -, /, 콤마 등을 사용할 수 있다.

콘솔입력에서 입력자료를 실수형 변수에 저장하려면 형식 지정자 %f 를 사용하며, 실수 double 형 변수에 저장하려면 형식지정자 %lf 를 사용한다.

문자 char 형 변수에 저장하려면 제어문자 %c 를 사용한다.

입력 scanf()에서는 저장될 자료형이 float 이면 %f 를, double 이면 %lf 로 구분하여 사용해야한다.

함수 getch()는 문자 하나를 입력하는 매크로 함수이고 putchar()는 반대로 출력하기 위한 매크로 함수이다.

이 함수를 이용하려면 printf()나 scanf()처럼 헤더파일 stdio.h 가 필요하다.

05.CHAPTER

연산자

연산식과 연산자

변수와 다양한 리터럴 상수 그리고 함수의 호출 등으로 구성되는 식을 연산식이라고 한다.

연산자는 산술연산자 +, -, * 기호와 같이 이미 정의된 연산을 수행하는 문자 또는 문자조합기호를 말한다. 연산에 참여하는 변수나 상수를 피연산자라고 한다.

연산식은 평가하여 항상 하나의 결과값을 갖는다.

연산자는 연산에 참여하는 피연산자의 개수에 따라 단항, 이항, 삼항 연산자로 나눌 수 있다.

삼항연산자는 조건연산자 ?: 가 유일하다.

++a 처럼 연산자가 앞에 있으면 전위연산자이며, a++과 같이 연산자가 뒤에 있으면 후위 연산자라고 한다.

산술연산자는 +, -, *, /, %로 각각 **더하기, 빼기, 곱하기, 나누기, 나머지** 연산자이다.

정수끼리의 나누기 연산결과는 소수부분을 버린 정수이다.

대입연산자 = 는 연산자 오른쪽의 연산값을 변수에 저장하는 연산자이다.

대입연산자의 왼쪽 부분에는 반드시 하나의 변수만이 올 수 있다. 이 하나의 변수를 왼쪽을 의미하는 left 단어에서 l-value 라 하며 오른쪽에 위치하는 연산식의 값을 오른쪽을 의미하는 right 단어에서 r-value 라고 한다.

대입연산식 $a = a + b$ 는 중복된 a 를 생략하고 간결하게 $a += b$ 로 쓸 수 있다.

산술연산자와 대입연산자를 이어붙인 연산자를 축약 대입연산자라고 한다.

증가연산자 ++과 감소연산자 --는 변수값을 각각 1 증가시키고, 1 감소시키는 기능을 수행한다.

증가연산자에서 $n++$ 와 같이 연산자 ++가 피연산자 n 보다 뒤에 위치하는 후위이면 1 증가되기 전 값이 연산 결과값이다. 반대로 $++n$ 과 같이 전위이면 1 증가된 값이 연산 결과값이다.

마찬가지로 감소연산자에서 $n--$ 와 같이 후위이면 1 감소되기 전 값이 연산 결과값이다.

반대로 $--n$ 이면 1 감소된 값이 연산 결과값이다.

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

관계연산자는 두 피연산자의 크기를 비교하기 위한 연산자이다.

관계연산자의 연산값은 비교결과가 참이면 0, 거짓이면 1 이다.

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, 0.0, w0 은 거짓을 의미하며, 0 이 아닌 모든 정수와 실수, 그리고 널 문자 'w0'가 아닌 모든 문자와 문자열은 모두 참을 의미한다.

논리연산자 &&와 ||는 피 연산자 두 개 중에서 왼쪽 피연산자만으로 논리연산 결과가 결정된다면 오른쪽 피연산자는 평가하지 않는다.

조건연산자는 조건에 따라 주어진 피연산자가 결과값이 되는 삼항연산자이다.

c 언어는 정수의 비트중심 연산자를 제공한다.

비트 논리 연산자는 피연산자 정수값을 비트 단위로 논리 연산을 수행하는 연산자이다.

비트 연산은 각 피연산자를 int 형으로 변환하여 연산하며 결과도 int 형이다.

AND 연산자인 &는 두 비트 모두 1 이어야 1 이며, OR 연산자인 |은 하나만 1 이어도 1 이고, ^는 서로 다르면 1 이고, 같으면 0 이다.

NOT 또는 보수 연산자인 ~은 단항연산자로 0 인 비트는 1 로, 1 인 비트는 0 으로 모두 바꾸는 연산자이다.

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

이동시 각 오른쪽 왼쪽에 빈자리가 생겨, 오른쪽 빈자리는 모두 0 으로 채워지며, 왼쪽 빈자리는 원래의 부호비트에 따라 0 또는 1 이 채워진다.

형변환 연산자와 연산자 우선순위

작은 범주의 INT 형에서 보다 큰 범주인 DOUBLE 형으로의 형 변환을 올림변환이라고 한다.

올림변환은 정보의 손실이 없으므로 컴파일러에 의해 자동으로 수행 될 수 있다.

컴파일러가 자동으로 수행하는 형변환을 묵시적 형변환이라 한다.

올림변환과 반대로 대입연산 `int a = 3.4` 에서 내림변환이 필요하다.

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

형변환 연산자'(type)피연산자'는 뒤에 나오는 피 연산자의 값을 괄호에서 지정한 자료형으로 변환하는 연산자이다.

형변환 연산자를 사용한 방식을 명시적 형변환이라고 한다.

-명시적 형변환 방법 -> `(int) 30.525`

상수나 변수의 정수값을 실수로 변환하려면 올림변환을 사용한다.

실수의 소수부분을 없애고 정수로 사용하려면 내림변환을 사용할 수 있다.

단항연산자인 형변환 연산자는 모든 이항연산자보다 먼저 계산한다.

연산자 `sizeof` 는 연산값 또는 자료형의 저장장소의 크기를 구하는 연산자이다. 연산자 `sizeof` 의 결과값은 바이트단위의 정수이다.

연산자 `sizeof` 는 피연산자가 `int` 와 같은 자료형인 경우 반드시 괄호를 사용해야한다.

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

06.CHAPTER

제어문

프로그램의 실행 흐름에서도 순차적인 실행뿐만 아니라 선택과 반복 등 순차적인 실행을 변형하여 프로그램의 실행 순서를 제어하는 제어문이 제공된다.

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

Ex) `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.

Ex) `for`, `while`, `do while`

분기 구문은 작업을 수행 도중 조건에 따라 반복이나 선택을 빠져나가거나, 일정구문을 실행하지 않고 다음 반복을 실행하거나, 지정된 위치로 이동하거나 또는 작업 수행을 마치고 이전 위치로 돌아가는 구문이다.

Ex) break, continue, goto, return

IF

문장 if 는 위에서 살펴본 조건에 따른 선택을 지원하는 구문이다.

형태는 if (cond) stmt;이다 if 문에서 조건식 cond 가 0 이 아니면 stmt 를 실행하고, 0 이면 stmt 를 실행하지 않는다.

문장 if 의 조건식은 반드시 괄호가 필요하며, 참이면 실행되는 문장은 반드시 들여쓰기를 하도록 한다.

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

조건문 if else 는 stmt1 과 stmt2 둘 중의 하나를 선택하는 구문이다.

정수 n 이 짝수인지 아니면 홀수인지 판단할 수 있는 조건식으로 $(n \% 2 == 0)$ 또는 $(n \% 2)$ 이 주로 사용될 수 있다.

SWITCH

switch 문을 사용하면 문장 if else 가 여러 번 반복되는 구문을 좀 더 간략하게 구현 할 수 있다.

switch 문은 주어진 연산식이 문자형 또는 정수형이라면 그 값에 따라 case 의 상수값과 일치하는 부분의 문장들을 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 중복 될 수 없다.

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.

일반적으로 switch 문에서 default 는 생략 될 수 있으며, 그 위치도 제한이 없다.

다만 default 를 위치시킨 이후에 다른 case 가 있다면 break 를 반드시 입력하도록 한다.

07.CHAPTER

반복문

WHILE

반복은 발 그대로 같거나 비슷한 일을 여러 번 수행하는 작업이다.

c 언어는 while, do while, for 세 가지 종류의 반복 구문을 제공한다.

반복조건을 만족하면 일정하게 반복되는 부분을 반복몸체라 한다.

문장 while (cond) stmt;는 반복조건인 cond를 평가하여 0이 아니면 반복몸체인 stmt를 실행하고 다시 반복조건 cond를 평가하여 while 문 종료 시 까지 반복한다.

반복이 실행되는 stmt를 반복몸체라고 부르며 필요하면 블록으로 구성 될 수 있다.

<while 문>

While (cond)

 Stmt;

Next;

반복횟수를 제어하는 변수를 제어변수라고 한다.

조건식 (cont <= 3)에서 상수 3은 최대 반복횟수를 지정하는 상수인 셈이다.

반복몸체에서 제어변수 count 횟수만큼 반복을 위해 count를 1 증가시키는 count++ 문장이 반드시 필요하다.

조건식 (count <= 3)은 4번 실행되며, 제어변수 count는 4이다.

while 문은 반복 전에 반복조건을 평가한다. Do while 문은 반복몸체 수행 후에 반복조건을 검사한다.

Do stmt; while(cond)는 가장 먼저 stmt를 실행한 이후 반복조건인 cond를 평가하여 0이 아니면 다시 반복몸체인 stmt;를 실행하고, 0이면 do while 문을 종료한다.

반복 횟수가 정해지지 않고 입력 받은 자료값에 따라 반복 수행의 여부를 결정하는 구문에 유용하다. 반복몸체에 특별히 분기 구문이 없는 경우, do while의 몸체는 적어도 한번은 실행되는 특징이 있다.

While 이후의 세미콜론은 반드시 필요하다는 것이다.

FOR

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

2개의 세미콜론은 반드시 필요하다.

반복조건 cond를 아예 제거하면 반복은 무한히 계속된다.

변수 i와 같이 반복의 횟수를 제어하는 변수를 제어변수라고 한다.

for 문은 주로 반복횟수를 제어하는 제어변수를 사용하며 초기화와 증감부분이 있는 반복문에 적합하다.

while 문은 반복횟수가 정해지지 않고 특정한 조건에 따라 반복을 결정하는 구문에 적합하다.

분기문

분기문은 정해진 부분으로 바로 실행을 이동하는 기능을 수행한다.

C 가 지원하는 분기문으로는 break, continue, goto, return 문이 있다.

반복내부에서 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 반복문체의 나머지 부분을 실행하지않고 다음 반복을 계속 유지하는 문장이다.

반복문 while 과 do while 반복 내부에서 continue 를 만나면 조건검사로 이동하여 실행한다.

반복문 for 문에서 continue 문을 만나면 증감부분으로 이동하여 다음 반복 실행을 계속한다.

Continue 이후의 문장은 실행되지 않고 뛰어 넘어간다는 것이다.

중첩된 반복에서의 continue 는 자신이 속한 가장 근접한 반복에서 다음 반복을 실행한다.

goto 문은 레이블이 위치한 다음 문장으로 실행순서를 이동하는 문장이다.

반복문에서 무한히 반복이 계속되는 것을 무한반복이라 한다.

While 과 do while 은 반복조건이 아예 없으면 오류가 발생하니 주의하도록 하자.

반복문 내부에 반복문이 또 있는 구문을 중첩된 반복문이라 한다.

08.CHAPTER

포인터

포인터 변수

메모리 공간은 8 비트인 1 바이트마다 고유한 주소가 있다.

메모리 주소는 0 부터 바이트마다 1 씩 증가한다. 메모리 주소는 저장장소인 변수이름과 함께 기억장소를 참조하는 또 다른 방법이다.

주소는 변수이름과 같이 저장장소를 참조하는 하나의 방법이다.

&가 피연산자인 변수의 메모리 주소를 반환하는 주소연산자이다.

변수의 주소값은 형식제어문자 %u 또는 %d 로 직접 출력할 수 있다.

그러나 최근 비주얼 스튜디오에서는 경고가 발생하니 주소값을 int 또는 unsigned 로 변환하여 출력한다. 만일 16 진수로 출력하려면 형식제어 문자 %p 를 사용한다. 주소연산자 &는 다음 사용에 주의가 필요하다.

변수의 주소도 포인터 변수에 저장할 수 있다.

변수의 주소값은 반드시 포인터 변수에 저장해야 한다.

포인터 변수는 주소값을 저장하는 변수로 일반 변수와 구별되며 선언방법이 다르다.

포인터 변수 선언에서 자료형과 포인터 변수 이름사이에 연산자 *를 삽입한다.

위 포인터 변수선언에서 보듯이 변수 자료형이 다르면 그 변수의 주소를 저장하는 포인터의 자료형도 달라야한다.

어느 변수의 주소값을 저장하려면 반드시 그 변수의 자료유형과 동일한 포인터 변수에 저장해야 한다. 포인터 변수선언에서 포인터를 의미하는 * 는 자료형과 변수이름 사이에만 위치하면 된다.

포인터 변수는 가리키는 변수의 종류에 관계없이 크기가 모두 4 바이트이다.

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

초기값을 대입하지 않으면 쓰레기 값이 들어가므로 포인터 변수에 지정할 특별한 초기값이 없는 경우에 0 번 주소 값인 NULL 로 초기값을 저장한다.

이 NULL 은 헤더파일 stdio.h 에 다음과 같이 정의되어 있는 포인터 상수로서 0 번지의 주소값을 의미한다.

자료유형(void*)는 아직 유보된 포인터이므로 모든 유형의 포인터 값을 저장할 수 있는 포인터 형이다.

포인터 변수가 갖는 주소로 그 주소의 원래 변수를 참조할 수 있다.

포인터 변수가 가리키고 있는 변수를 참조하려면 간접연산자 * 를 사용한다.

포인터 pprint 가 가리키는 변수가 data 라면 *pprint 는 변수 data 를 의미한다.

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

더하기와 빼기 연산에는 포인터 변수가 피연산자로 참여할 수 있다.

포인터 변수의 주소값을 갖는 변수를 이중 포인터라 한다.

포인터의 포인터를 모두 다중 포인터라고 한다.

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

09.CHAPTER

배열

배열

배열은 여러 변수들이 같은 배열 이름으로 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 자료구조이다.

배열은 한 자료유형의 저장공간인 원소를 동일한 크기로 지정된 배열크기만큼 확보한 연속된 저장공간이다. 배열에서 중요한 요소는 배열이름, 원소 자료유형, 배열 크기이다.

배열 원소는 첨자 번호라는 숫자를 이용해 쉽게 접근할 수 있다.

배열선언

배열 선언은 `int data[10];` 과 같이 원소자료유형 배열이름[배열크기];로 한다.

배열선언 시 초기값 지정이 없다면 반드시 배열크기는 양의 정수로 명시되어야 한다.

배열의 크기를 지정하는 부분에는 양수 정수로 리터럴 상수와 매크로 상수 또는 이들의 연산식이 올 수 있다. 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.

배열선언 시 대괄호 안의 수는 배열 크기이다. 그러나 선언 이후 대괄호 안의 수는 원소를 참조하는 번호인 첨자라는 것을 명심하자.

첨자의 시작이 0 이므로 배열의 순번은 첨자보다 1 이 크다는 것을 명심해야한다.

또한 `int score[5];` 라면 배열 크기가 5 이므로 첨자는 0 에서 4 까지 유효하다.

배열을 함수내부에서 선언 후 원소에 초기값을 저장하지않으면 쓰레기값이 저장되므로 항상 초기값을 저장해야한다.

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.

일반 배열선언과 다르게 배열크기는 생략할 수 있으며, 생략하면 자동으로 중괄호 사이에 기술된 원소 수가 배열크기가 된다.

원소값을 나열하기 위해 콤마를 사용하고 전체를 중괄호로 묶는다.

Ex) `int grade[] = {98, 88, 92, 95};`

만일 배열크기가 초기값 원소 수보다 크면 지정하지 않은 원소의 초기값은 자동으로 모두 기본값으로 저장된다.

정수형은 0, 실수형은 0.0 문자형은 'w0'인 널 문자가 자동으로 채워진다.

배열크기가 초기값 원소 수보다 작으면 배열 저장공간을 벗어나므로 "이니셜라이저가 너무 많습니다." 라는 문법오류가 발생한다.

이차원 배열

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

이차원 배열선언은 2 개의 대괄호가 필요하다.

첫번째 대괄호에는 배열의 행 크기, 두번째는 배열의 열 크기를 지정한다.

배열선언 시 초기값을 저장하지않으면 반드시 행과 열의 크기는 명시되어야 한다.

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 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 첫번째 행 모든 원소가 메모리에 할당된 이후에 두번째 행의 원소가 순차적으로 할당된다. C 언어와 같은 배열의 이러한 특징을 행 우선 배열이라 한다.

외부반복 제어변수 `i`는 행을 0 에서 (행의 수 -1)까지 순차적으로 참조하며, 내부반복 제어변수 `j`는 0 에서 (열의 수-1)까지 열을 순차적으로 참조한다.

배열원소의 저장값이 행과 열의 관계식으로 만들 수 있다면 출력과 같이 중복된 반복문을 사용하여 값을 저장할 수 있다.

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

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

이차원 배열선언 초기값 지정에도 첫번째 대괄호 내부의 행의 크기는 명시하지 않을 수 있다.

두번째 대괄호 내부의 열의 크기는 반드시 명시해야 한다.

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

배열과 포인터

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

배열이름 score 는 배열 첫번째 원소의 주소를 나타내는 상수로 &score[0]와 같으며 간접연산자를 이용한 *score 는 변수 score[0]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)으로 포인터 p 가 가리키는 값을 1 증가시킨 후 참조한다.

포인터 변수는 동일한 자료형끼리만 대입이 가능하다.

만일 대입문에서 포인터의 자료형이 다르면 경고가 발생한다.

*pi 로 수행하는 간접참조는 pi 가 가리키는 주소에서부터 4 바이트 크기의 int 형 자료를 참조한다는 것을 의미한다.

동일한 메모리의 내용과 주소로부터 참조하는 값이 포인터의 자료형에 따라 달라진다.

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

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

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

10.CHAPTER

함수

함수정의와 호출

특정한 작업을 처리하도록 작성한 프로그램 단위를 함수라고한다.

함수는 필요한 입력을 받아 원하는 어떤 기능을 수행한 후 결과를 반환하는 프로그램 단위이다.

함수는 라이브러리 함수와 사용자 정의함수로 구분 할 수 있다.

사용자가 직접 개발한 함수를 사용하기 위해서는 함수선언, 함수호출, 함수정의가 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 프로그래밍 방식을 절차적 프로그래밍 방식이라 한다.

함수정의는 함수머리와 함수몸체로 구성된다.

함수머리는 반환형과 함수이름, 매개변수 목록으로 구성된다.

함수몸체에서는 함수가 수행해야 할 문장들로 구성된다.

함수가 반환값이 없다면 반환형으로 void 를 기술한다.

return 문장은 함수에서 반환값을 전달하는 목적과 함께 함수의 작업 종료를 알리는 문장이다.

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.

매개변수

함수 매개변수는 함수를 호출하는 부분에서 함수몸체로 값을 전달할 목적으로 이용된다.

함수정의에서 매개변수는 필요한 경우 자료형과 변수명의 목록으로 나타내며 필요 없으면 키워드 `void` 를 기술한다.

함수정의에서 기술되는 매개변수 목록의 변수를 형식매개변수라 한다.

형식매개변수는 함수 내부에서만 사용할 수 있는 변수이다. 함수의 매개변수로 배열을 전달한다면 한 번에 여러 개의 변수를 전달하는 효과를 가져온다.

함수 `sum()`은 실수형 배열의 모든 원소의 합을 구하여 반환하는 함수이다.

함수헤더에 `int ary[]`로 기술하는 것은 `int *ary`로도 대체 가능하다.

다차원 배열을 인자로 이용하는 경우, 함수원형과 함수정의의 헤더에서 첫 번째 대괄호 내부의 크기를 제외한 다른 모든 크기는 반드시 기술되어야 한다.

재귀와 함수 구현

함수구현에서 자신 함수를 호출하는 함수를 재귀함수라 한다.

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

특정한 나열 순서나 규칙을 가지지 않는 연속적인 임의의 수를 난수라 한다.

함수 `rand()`의 함수원형은 헤더파일 `stdlib.h`에 정의되어 있다.

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

문자 관련 함수는 헤더파일 `ctype.h`에 매크로로 정의되어 있다.

11.CHAPTER

문자와 문자열

문자는 영어의 알파벳이나 한글의 한 글자를 작은 따옴표로 둘러싸서 'A'와 같이 표기하며 작은 따옴표에 의해 표기된 문자를 문자 상수라고 한다.

문자의 모임인 일련의 문자를 문자열(string)이라 한다.

문자열은 일련의 문자 앞 뒤로 큰 따옴표로 둘러싸서 "java"로 표기한다.

문자의 나열인 문자열은 'ABC' 처럼 작은 따옴표로 둘러싸도 문자가 될 수 없으며 오류가 발생한다.

C 언어에서 char 형 변수에 문자를 저장한다.

문자열을 저장하려면 문자의 모임인 '문자 배열'을 사용한다.

배열 초기화 시 배열크기는 지정하지 않는 것이 더 편리하며, 만일 지정한다면 마지막 문자인 'W0'을 고려해 실제 문자 수보다 1 이 더 크게 배열크기를 지정해야한다.

만일 지정한 배열크기가 (문자수+1)보다 크면 나머지 부분은 모두 'W0'문자로 채워진다.

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

문자 입력을 위한 함수 getchar()는 라인 버퍼링방식을 사용한다.

Getche()는 버퍼를 사용하지않고 문자 하나를 바로바로 입력할 수 있는 함수이다.

함수를 이용하려면 헤더파일 conio.h 를 삽입해야한다.

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

함수 gets()는 한 행의 문자열 입력에 유용한 함수이다.

함수 puts()는 한 행에 문자열을 출력하는 함수이다.

함수 printf()와 scanf()는 다양한 입출력에 적합하고 문자열 입출력 함수 puts()와 gets()는 처리 속도가 빠르다는 장점이 있다.

문자열 비교와 복사, 그리고 문자열 연결 등과 같은 다양한 문자열 처리는 헤더파일 string.h 에 함수원형으로 선언된 라이브러리 함수로 제공된다.

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 앞 문자열에 뒤 문자열의 null 문자까지 연결하여, 앞의 문자열 주소를 반환하는 함수이다.

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

문장 ptoken = strtok(str, delimiter);으로 첫 토큰을 추출한다.

결과를 저장한 ptoken 이 NULL 이면 더 이상 분리할 토큰이 없는 경우이다.

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

여러 문자열 처리

여러 개의 문자열을 처리하는 하나의 방법은 문자 포인터 배열을 이용하는 방법이다.

또 다른 방법은 문자의 이차원 배열을 이용하는 방법이다.

12.CHAPTER

변수 유효범위

전역변수와 지역변수

변수의 참조가 유효한 범위를 변수의 유효 범위라고 한다.

변수의 유효범위는 크게 지역 유효 범위와 전역 유효 범위로 나눌 수 있다.

지역 유효범위는 함수 또는 블록 내부에서 선언되어 그 지역에서 변수의 참조가 가능한 범위이다.

지역변수는 함수 또는 블록에서 선언된 변수이다.

함수나 블록에서 지역변수는 선언 문장 이후에 함수나 블록의 내부에서만 사용이 가능하다.

함수의 매개변수도 함수 전체에서 사용 가능한 지역변수와 같다.

지역변수는 선언 후 초기화하지 않으면 쓰레기값이 저장되므로 주의해야 한다.

지역변수가 할당되는 메모리 영역을 스택이라 한다.

지역변수는 선언된 부분에서 자동으로 생성되고 함수나 블록이 종료되는 순간 메모리에서 자동으로 제거된다.

전역변수는 함수 외부에서 선언되는 변수이다. 전역변수는 외부변수라고도 부른다.

전역변수는 일반적으로 프로젝트의 모든 함수나 블록에서 참조할 수 있다.

전역 변수에 예상하지 못한 값이 저장된다면 프로그램 어느 부분에서 수정되었는지 알기 어려운 단점이 있다.

정적 변수와 레지스터 변수

4 가지의 기억부류인 auto, register, static, extern 에 따라 할당되는 메모리 영역이 결정되고 메모리의 할당과 제거 시기가 결정된다.

기억 부류 auto 와 register 는 지역변수에만 이용이 가능하고 static 은 지역과 전역 모든 변수에 이용 가능하다. Extern 은 전역변수에만 이용 가능하다.

키워드 extern 을 제외하고 나머지 3 개의 기억부류의 변수선언에서 초기값을 저장할 수 있다.

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

레지스터 변수는 키워드 register 를 자료형 앞에 넣어 선언한다.

레지스터 변수는 일반 메모리에 할당되는 변수가 아니므로 주소 연산자 &를 사용할 수 없다.

주로 레지스터 변수는 처리 속도를 증가시키려는 변수에 이용한다.

특히 반복문의 횟수를 제어하는 제어변수에 이용하면 효과적이다.

변수 선언에서 자료형 앞에 키워드 static 을 넣어 정적변수를 선언할 수 있다.

정적변수는 초기값을 지정하지않으면 자동으로 자료형에 따라 0 이나 'W0'또는 NULL 값이 저장된다.

함수나 블록에서 정적으로 선언되는 변수가 정적 지역변수이다.

정적 지역변수는 함수나 블록을 종료해도 메모리에서 제거되지않고 계속 메모리에 유지 관리되는 특성이 있다.

함수 외부에서 정적으로 선언되는 변수가 정적 전역변수이다.

정적 전역변수는 선언된 파일 내부에서만 참조가 가능한 변수이다.

전역변수는 프로그램이 크고 복잡하면 전역변수의 사용은 원하지 않는 전역변수의 수정과 같은 부작용의 위험성이 항상 존재한다.

메모리 영역과 변수 이용

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

힙 영역은 동적 할당되는 변수가 할당되는 저장공간이다.

스택영역은 함수 호출에 의한 형식 매개변수 그리고 함수 내부의 지역변수가 할당되는 저장공간이다.

스택영역은 메모리 주소가 높은 값에서 낮은 값으로 저장 장소가 할당된다.

그러므로 함수 호출과 종료에 따라 높은 주소에서 낮은 주소로 메모리가 할당되었다가 다시 제거되는 작업이 반복된다.

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

해당파일 내부에서만 변수를 공유하고자 하는 경우는 정적 전역변수를 이용한다.

13.CHAPTER

구조체와 공용체

구조체는 정수나 문자, 실수나 포인터 그리고 이들의 배열 등을 묶어 하나의 자료형으로 이용하는 것이다.

즉 연관성이 있는 서로 다른 개별적인 자료형의 변수들을 하나의 단위로 묶은 새로운 자료형을 구조체라고 한다. 구조체는 연관된 멤버로 구성되는 통합 자료형으로 대표적인 유도 자료형이다.

기존 자료형으로 새로이 만들어진 자료형을 유도 자료형이라 한다.

구조체를 자료형으로 사용하려면 먼저 구조체를 정의해야 한다. 먼저 구조체를 만들 구조체 틀을 정의해야한다.

구조체를 정의하는 방법은 키워드 struct 다음에 구조체 태그이름을 기술하고 중괄호를 이용하여 원하는 멤버를 여러 개의 변수로 선언하는 구조다. 구조체를 구성하는 하나 하나의 항목을 구조체 멤버 또는 필드라 한다.

구조체 정의는 변수의 선언과는 다른 것으로 변수선언에서 이용될 새로운 구조체 자료형을 정의하는 구문이다.

구조체 멤버로는 일반 변수, 포인터 변수, 배열, 다른 구조체 변수 및 구조체 포인터도 허용된다.

새로운 자료형 struct account 형 변수 mine 을 선언하려면 struct account mine; 으로 선언한다.

초기화 값은 다음과 같이 중괄호 내부에서 구조체의 각 멤버 정의 순서대로 초기값을 쉼표로 구분하여 기술한다. 배열과 같이 초기값에 기술되지 않은 멤버값은 자료형에 따라 기본값인 0, 0.0, 'W0' 등으로 저장된다.

선언된 구조체형 변수는 접근 연산자 . 를 사용하여 멤버를 참조할 수 있다.

공용체는 서로 다른 자료형의 값을 동일한 저장공간에 저장하는 자료형이다.

공용체 변수의 크기는 멤버 중 가장 큰 자료형의 크기로 정해진다.

공용체의 멤버는 모든 멤버가 동일한 저장공간을 사용하므로 동시에 여러 멤버의 값을 동시에 저장하여 이용할 수 없으며, 마지막에 저장된 단 하나의 멤버 자료값만을 저장한다.

공용체의 초기화 값은 공용체 정의 시 처음 선언한 멤버의 초기값으로만 저장이 가능하다.

공용체 변수로 멤버를 접근하기 위해서는 구조체와 같이 접근연산자 .를 사용한다.

자료형 재정의

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

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

문장 typedef 도 일반 변수와 같이 그 사용 범위를 제한한다.

구조체 struct date 가 정의된 상태에서 typedef 사용하여 구조체 struct date 를 date 로 재정의할 수 있다.

구조체 포인터

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

구조체 포인터 멤버 접근연산자 ->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.

다른 배열과 같이 동일한 구조체 변수가 여러 개 필요하면 구조체 배열을 선언하여 이용할 수 있다.