

C Perfect

포트폴리오

컴퓨터정보공학과

20193429

이현복

목차

머리말 포트폴리오 제작! 3p

Chapter 1 프로그래밍언어 개요 3p - 4p

Chapter 2 C프로그래밍 첫걸음 5p -7p

Chapter 3 자료형과 변수 7p - 13p

Chapter 4 전처리와 입출력 13p - 15p

Chapter 5 연산자 16p - 18p

Chapter 6 조건과반복 19p - 21p

Chapter 7 포인터 기초 21p - 23p

Chapter 8 배열 24p - 27p

Chapter 9 함수 기초 27p - 32p

Chapter 10 문자와 문자열 32p - 34p

Chapter 11 변수 유효범위 34p - 37p

Chapter 12 구조체와 공용체 37p - 43p

꼬리말 포트폴리오를 마치며... 43p

머리말 포트폴리오 제작!

중간고사를 대체하는 포트폴리오를 작성하면서 과제를 한다는 느낌이 아닌 지금까지 배운 내용을 다시 되짚어 본다는 느낌으로 작성하려고 합니다. 1단원부터 차근차근 중요하다고 생각이 드는 내용과 처음 배울 때는 어려워 기피하고 넘어갔던 내용들 위주로 이 포트폴리오를 작성해 나가겠습니다. 감사합니다.

01 프로그래밍언어 개요

프로그램

컴퓨터와 스마트폰에서 특정 목적의 작업을 수행하기 위한 관련 파일의 모임을 **프로그램**이라 한다.

이 모든기기에서 사용되는 프로그램을 만드는 사람을 **프로그래머**라 한다.

프로그램을 개발하기 위해 사용하는 언어가 **프로그래밍 언어**이다. 프로그래밍 언어는 사람이 컴퓨터에게 지시할 명령어를 기술하기 위하여 만들어진 언어이다.

C 언어의 특징

1. 절차지향 언어

절차지향언어란 시간의 흐름에 따라 정해진 절차를 실행한다는 의미로 C언어는 문제의 해결 순서와 절차의 표현과 해결이 쉽도록 설계된 프로그램 언어이다.

2. 처음에 다소 어려움

문법이 상대적으로 간결한 대신 많은 내용을 함축하고 있으며, 비트와 포인터의 개념, 메모리 할당과 해제 등의 관리로 실제 C언어는 어렵게 사실이다. 하지만 C언어는 다른 프로그램에 많은 영향을 끼쳤으므로 한번 익혀두면 다른 프로그래밍 언어 습득에도 많은 도움을 준다.

3. 뛰어난 이식성

C언어는 다양한 CPU와 플랫폼의 컴파일러를 지원하기 때문에 이식성이 좋다.

4. 간결하고 효율적인 언어

C언어는 다양한 연산과 이미 개발된 시스템 라이브러리를 제공하며, 함수의 재귀 호출, 포인터와 메모리 관리 기능 세세한 부분까지 제어할 수 있고, C언어로 작성된 프로

그럼은 크기도 작으며, 메모리도 효율적으로 활용해 실행속도가 빠르다는 장점이 있다.

소프트웨어와 알고리즘

소프트웨어는 보통 '프로그램' 이라고 부르는 것 외에도 데이터와 문서까지를 포괄하는 개념이다.

알고리즘이란 어떠한 문제를 해결하기 위한 절차나 방법으로 명확히 정의된 유한 개의 규칙과 절차의 모임이다. **컴퓨터 프로그램**은 특정한 업무를 수행하기 위한 정교한 알고리즘들의 집합이라고 간주할 수 있다.

소프트웨어 개발 과정

요구사항 분석 > 설계 > 구현 > 검증 > 유지보수

1단계 **요구사항 분석**은 사용자의 요구사항을 파악하여 분석하는 단계이다.

2단계 **설계**는 프로그래머가 알고리즘을 이용 해 소프트웨어를 설계한다.

3단계 **구현**은 2단계에 설계된 내용에 따라 코딩한다.

4단계 **검증**은 작성된 프로그램을 테스트한다.

5단계 **유지보수**는 프로그램을 문서화하고 유지 보수한다.

02 C 프로그래밍 첫걸음

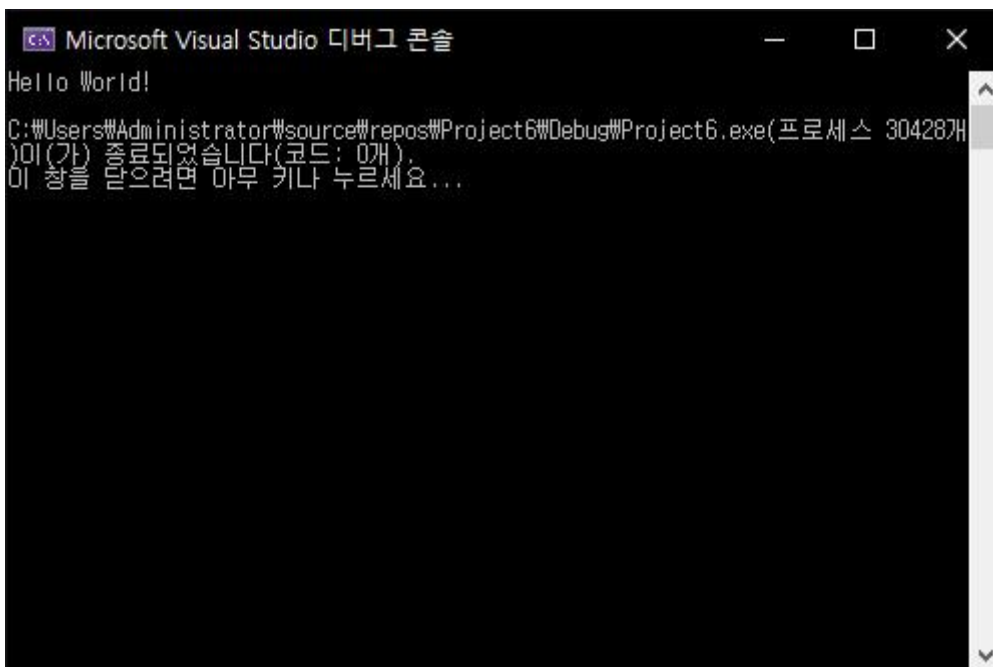
```
1  #include <stdio.h>
2
3  int main()
4  {
5      puts("Hello World!");
6
7      return 0;
8  }
```

프로그래머라면 거쳐야 하는 코딩을 한번 해봤다!

puts() : 매개변수로 입력된 문자열을 출력하는 문장이다

이 puts()에 "Hello World!"를 매개변수로 넣어 이 문자열을 출력했다.

결과값



```
Microsoft Visual Studio 디버그 콘솔
Hello World!
C:\Users\Administrator\source\repos\Project6\Debug\Project6.exe(프로세스 30428개)
)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

함수의 이해

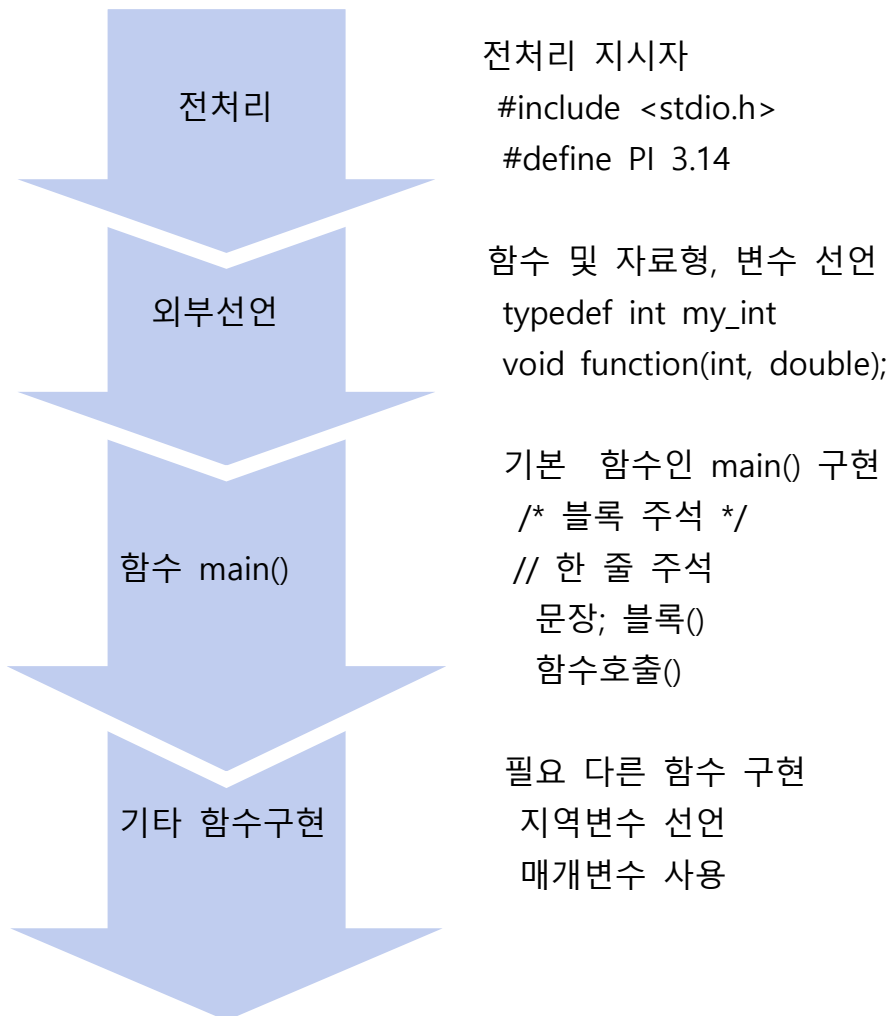
C언어는 함수로 구성된다. 함수는 '입력'(매개변수)은 여러 개 사용 될 수 있지만, 결과 값은(리턴값) 꼭 하나여야 한다.

함수는 프로그래머가 직접 만드는 **사용자 정의 함수**와 시스템이 미리 만들어 놓은 함수를 **라이브러리 함수**라 한다.

프로그래밍 기초

프로그램 구조

C 프로그램 소스의 구조



한 프로젝트는 단 하나의 함수 `main()`과 다른 함수로 구현되며, 최종적으로 프로젝트 이름으로 **하나의 실행 파일**이 만들어진다.

C 프로그램은 적어도 **`main()` 함수 하나는 구현되어야** 응용 프로그램으로 실행될 수 있다.

키워드

문법적으로 고유한 의미를 갖는 예약된 단어가 있는데 이 단어들을 다른 용도로 사용해서는 안된다. 이 예약된 단어들을 **키워드**라고 한다.

ex) if, int, short, for, float, else, void, char

식별자

프로그래머가 마음대로 정의해서 사용하는 단어를 **식별자**라고 한다. 이 식별자를 사용하는 데에는 사용규칙이 있다.

- 식별자는 키워드와 철자,대문자,소문자 등 무엇이든 달라야 한다.
- 식별자는 영문자,숫자,밑줄로 구성되며 첫 문자로 숫자가 나올 수 없다.
- 프로그램 내부의 일정한 영역에서는 서로 구별되어야 한다.
- 키워드는 식별자로 이용할 수 없다.
- 식별자는 대소문자를 모두 구별한다. Ace, ACE, AcE 모두 다른 변수이다.
- 식별자의 중간에 공백이 들어갈 수 없다.

문장

프로그래밍 언어에서 컴퓨터에게 명령을 내리는 최소 단위를 문장이라 하며, 문장은 마지막에 세미콜론 ;으로 종료된다. 여러 개의 문장을 묶으면 블록이라고 하며, { } 중괄호로 열고 닫는다.

주석

일반 문장과 달리 프로그램에 전혀 영향을 미치지 않는 설명문을 주석이라고 한다. 주석이 중요한 이유는 자신이나 타인이 작성한 코딩을 이해하기 쉽게 하기 때문이다. 그러므로 주석은 이 소스를 보는 모든 사람이 이해할 수 있도록 도움이 되는 설명을 담고 있어야 한다.

주석은 2가지 방법으로 처리 되는데 한줄 주석 // 과 블록 주석 /* */을 이용한다.

03 자료형과 변수

자료형

자료형은 프로그래밍 언어에서 자료를 식별하는 종류를 말한다.

변수

정수와 실수, 문자 등의 자료값을 저장하는 공간을 변수라 한다. 저장되는 값에 따라 변수값은 바뀔 수 있으며 마지막에 저장된 **하나의 값만 저장 유지된다.**

변수선언

변수선언은 컴파일러에게 프로그램에서 사용할 저장 공간인 변수를 알리는 역할이다.

변수 초기화

변수선언만 하고 아무내용도 저장하지 않으면 쓰레기 값이 저장되며, 오류가 발생한다. 그러므로 변수를 선언한 이후에는 반드시 값을 저장하여야 하는데, 이를 변수의 초기화라 한다.

```
int age; // 변수선언
```

```
age = 14; // 변수 초기화
```

```
int age = 14; //변수 선언과 동시에 초기화
```

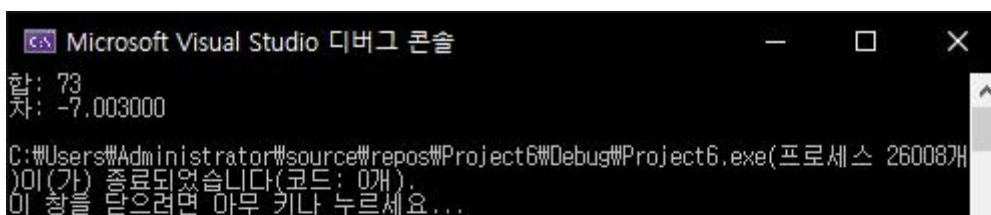
변수의 3요소

변수의 자료형, 변수의 이름, 변수 저장 값을 **변수의 3요소**라 한다. 변수의 3요소중 변수의 자료형, 이름은 바뀌지 않으나, 변수의 저장값은 대입문장에 의해 바뀔 수 있다. 저장 값이 계속 바뀔 수 있으므로 변수라 하는 것이다.

지금까지 배운 내용으로 간단한 예제를 풀어봤다.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int a = 30, b = 43; // int형 변수 a,b 선언과 동시에 초기화
6      int sum; // a+b 를 위한 변수 선언
7      sum = a + b; // sum에 a+b 대입
8
9      double x = 38.342, y = 45.345; // double형 변수 x,y 선언과 동시에 초기화
10     double x_y; // x-y 를 위한 변수 선언
11     x_y = x - y; // x_y에 x-y 대입
12
13     printf("합: %d\n", sum ); // sum을 이용해 a+b의 값 출력
14     printf("차: %f\n", x_y ); // x_y를 이용해 x-y의 값 출력
15
16
17
18     return 0;
19 }
```

결과 값



```
Microsoft Visual Studio 디버그 콘솔
합: 73
차: -7.003000
C:\Users\Administrator\source\repos\Project6\Debug\Project6.exe (프로세스 26008개)
(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```


정수 자료형

정수형 int

정수형의 기본 키워드는 int이다. int에서 파생된 자료형 short와 long이 있다. 단어의 뜻처럼 short는 작은 값을, long은 큰 값을 저장한다. 사용 범위에 따라 적절하게 short와 long 자료형을 선택해야 한다.

unsigned 자료형

0과 양수만을 처리하는 정수 자료형은 short, int, long 앞에 키워드 unsigned를 표시한다. 부호가 없는 정수인 unsigned int는 0과 양수만을 저장할 수 있는 자료형이다. int는 생략이 가능하므로 unsigned int와 unsigned는 같은 자료형이다.

정수 자료형의 표현 범위

음수지원 여부	자료형	크기	표현 범위
부호 있는 정수형 signed	signed short	2 바이트	-32,768 ~ 32,767
	signed int	4 바이트	-2,147,483,648 ~ 2,147,483,647
	signed long	4 바이트	-2,147,483,648 ~ 2,147,483,647
부호 없는 정수형 unsigned	unsigned short	2 바이트	0 ~ 65535
	unsigned int	4 바이트	0 ~ 4,294,967,295
	unsigned long	4 바이트	0 ~ 4,294,967,295

정수 자료형을 활용한 예제를 만들어 봤다.

```
#include <stdio.h>

int main(void)
{
    short s1 = 5000; // -32767 ~ 32767 까지
    int i1 = -1500000000; // -2,147,483,648 ~ 2,147,483,647 까지

    unsigned short int us1 = 60000; // 0 ~ 65535까지
    unsigned int ui1 = 4000000000; // 0 ~ 4,294,967,295 까지
    //int는 생략가능하다.

    printf("저장값: %d %d\n", s1, i1);
    printf("저장값: %u %u\n", us1, ui1);

    long long int l1l1 = 4500000000000000;
    // 64비트 운영체제부터 사용가능하고 약 922경 정도의 수를 음수와 양수로 지원한다. _int64 로도 사용가능하다.

    printf("저장값: %lld\n", l1l1);
}
```

결과값

```

Microsoft Visual Studio 디버그 콘솔
저장값: 5000 -1500000000
저장값: 60000 4000000000
저장값: 4500000000000000000
C:\Users\Administrator\source\repos\Project6\Debug\Project1.exe(프로세스 28904개)
)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...

```

부동소수 자료형

3.14, 5.124과 같은 실수를 표현하는 자료형이다. 키워드는 **float, double, long double** 세가지이다. float는 4바이트이며, double과 long double은 모두 8바이트이다. double과 long double은 사실상 같다.

자료형	크기	정수의 유효자릿수	표현 범위
float	4 바이트	6~7	1.175494351E-38F에서 3.402823466E+38F까지
double	8 바이트	15~16	2.2250738585072014E-308에서 1.7976931348623158E+308까지
long double	8 바이트	15~16	2.2250738585072014E-308에서 1.7976931348623158E+308까지

일반적으로 3.14같은 표현은 모두 double로 인식하기 때문에 float형 변수에 저장하려면 3.14F와 같이 float형 상수로 저장해야 한다. 3.14로 하면 오류가 발생한다.

문자형 자료형

문자형 자료형은 char, signed char, unsigned char 세가지가 있다. 문자형의 저장공간 크기는 모두 1바이트이다.

자료형	크기	표현 범위
char	1 바이트	-128에서 127까지 (문자는 0에서 127까지 사용)
signed char	1 바이트	-128에서 127까지
unsigned char	1 바이트	0에서 255까지

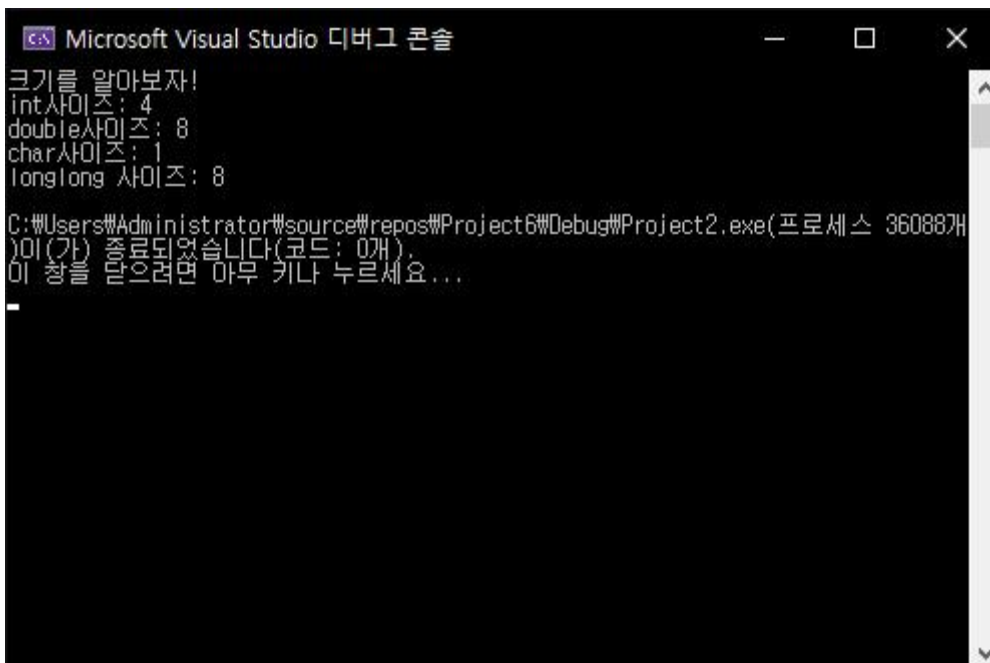
아스키 코드

C에서 문자형에 실제로 저장되는 값은 정수형이며, 이 정수는 아스키 코드 표에 의한 값이다. 아스키 코드는 총 127개의 문자로 구성된다. 아스키 코드에서 0~31, 127번 문자는 인쇄할 수 없는 문자이다. 32~126번 문자는 인쇄할 수 있는 인쇄문자이다.

지금까지 배운 자료형들의 크기를 알아보고 싶다면 연산자 `sizeof`를 이용하면 자료형, 변수, 상수의 **저장공간 크기를 바이트 단위로** 알 수 있다. 이 `sizeof`를 이용한 예제를 만들어보자.

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5
6      int intsize = sizeof(int); // int형의 크기를 바이트 단위로 나타냄.
7      int doublesize = sizeof(double); // double형의 크기를 바이트 단위로 나타냄.
8
9      printf("크기를 알아보자!\n");
10
11     printf("int사이즈: %d\n", intsize); // intsize 출력
12     printf("double사이즈: %d\n", doublesize); // doublesize 출력
13     printf("char사이즈: %d\n", sizeof(char)); // 변수에 저장하지 않고 바로 char의 크기를 출력
14     printf("longlong 사이즈: %d\n", sizeof(1LL)); // 1을 long long 형태로 저장 후 크기 출력
15
16 }
17
```

결과값



```
Microsoft Visual Studio 디버그 콘솔
크기를 알아보자!
int사이즈: 4
double사이즈: 8
char사이즈: 1
longlong 사이즈: 8

C:\Users\Administrator\source\repos\Project6\Debug\Project2.exe(프로세스 36088개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

오버플로와 언더플로

자료형의 범주에서 벗어난 값을 저장하면 오버플로 또는 언더플로가 발생한다.

정수형 자료형에서 최대값+1은 오버플로로 인해 최소값이 되고 최소값-1은 언더플로로 최대값이 된다 이 특징을 **정수의 순환**이라고한다.

상수의 개념과 표현방법

상수는 이름 없이 있는 그대로 표현한 자료값이나 이름이 있으나 하나의 값만으로 사용되는 자료값을 말한다. 상수는 크게 분리하면 **리터럴 상수**와 **심볼릭 상수**로 구분된다.

리터럴 상수는 이름이 없이 소스에 그대로 표현해 의미가 전달되는 다양한 자료값이고, 심볼릭 상수는 변수처럼 이름을 갖는 상수를 말한다. 이러한 심볼릭 상수를 표현하는 방법은 **const 상수**, **매크로 상수**, **열거형 상수**를 이용하는 세가지 방법이 있다.

구분	표현 방법	설명	예
리터럴 상수 (이름 X)	정수형 실수형 문자 문자열 상수	다양한 상수를 있는 그대로 기술	32, "C 언어", 100L, 'A', 025
심볼릭 상수 (이름 O)	const 상수	키워드 const를 이용한 변수 선언과 같으며, 수정할 수 없는 변수 이름으로 상수 정의	const double PI = 3.141592;
	매크로 상수	전처리기 명령어 #define으로 다양한 형태를 정의	#define PI 3.141592
	열거형 상수	정수 상수 목록 정의	enum bool {FALSE, TRUE};

const 상수

변수 선언시 자료형 또는 변수 앞에 키워드 const가 놓이면 이 변수는 심볼릭 상수가 된다. 상수는 변수선언 시 반드시 초기값을 저장해야 한다.

ex) int const FIVE = 5; const int SEVEN = 7;
const의 위치는 자료형의 앞 뒤 둘 다 가능하다.

열거형 상수

키워드 enum을 사용하여 정수형 상수 목록 집합을 정의하는 자료형이다. 열거형 상

수에서 목록 첫 상수의 기본값이 0이며 다음부터 1씩 증가하는 방식으로 상수값이 자동으로 부여된다. 상수 목록에서 특정한 정수를 지정 할 수 있는데 따로 지정한 상수의 뒷 상수는 지정한 상수부터 1씩 증가한 상수값으로 정의가 된다.

ex) enum DAY {SUN, MON, TUE, WED, THU = 14, FRI, SAT};
 순서대로 0, 1, 2, 3, 14, 15, 16으로 저장된다.

매크로상수

전처리 지시자 #define은 매크로 상수로 정의하는 지시자이다. 일반 변수와 구분하기 위해 #define 상수도 주로 대문자 이름으로 정의하는데 이를 **매크로 상수**라고 한다.

04 전처리

전처리지시자

컴파일러가 컴파일 하기 전에 전처리의 전처리 과정이 필요하다. 전처리 과정에서 처리되는 문장을 **전처리 지시자** 라고 한다. 헤더파일은 전처리 지시자인 #include, #define 등과 앞으로 배울 자료형의 재정의, 함수원형 정의 등과 같은 문장이 있는 텍스트 파일이다. 대표적인 헤더파일인 stdio.h는 printf(), scanf(), putchar(), getchar() 등과 같은 입출력 함수를 위한 함수원형 등의 정의된 헤더파일이다.

형식지정자

형식문자 종류 표

서식문자	자료형	출력 양식
c	char, int	문자 출력
d, i	int	부호 있는 정수 출력으로, ld는 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 십진수로 출력
e, E	double	기본으로 m.ddddddExxx의 지수 형식 출력 123456.789면 1.234567e+005로 출력
f, lf	double	소수 형식 출력으로 m.123456처럼 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 E를 사용

s	char *	문자열에서 'wo'가 나올 때 까지 출력되거나 정밀도에 의해 주어진 문자 수만큼 출력
p	void *	주소값을 십육진수 형태로 출력
%		%를 출력

입력함수 scanf()

scanf()는 대표적인 입력함수이고 %s와 %d같은 형식지정자를 사용한다.

ex) scanf("%d, %d", &a, &b);

&는 주소연산자로 뒤에 표시된 피연산자인 변수 주소값이 연산값으로,scanf()의 입력 변수목록에는 키보드에 입력값이 저장되는 변수를 찾는다는 의미에서 반드시 변수의 주소연산식 '&변수이름'이 인자로 사용되어야한다.

scanf()를 이용한 예제를 만들어 보자.

```

1
2 #define _CRT_SECURE_NO_WARNINGS // scanf() 오류 방지를 위한 상수 정의
3
4 #include <stdio.h>
5
6 int main(void)
7 {
8     int age;
9
10    printf("당신은 몇살인가요? ");
11
12    scanf("%d", &age); //변수에 주소연산자 & 입력
13
14    printf("제 나이는 %d 입니다.", age);
15
16
17 }
```

결과값

```

Microsoft Visual Studio 디버그 콘솔
당신은 몇살인가요? 24
제 나이는 24 입니다.
C:\Users\Administrator\source\repos\Project6\Debug\Project3.exe(프로세스 16512개)
이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...

```

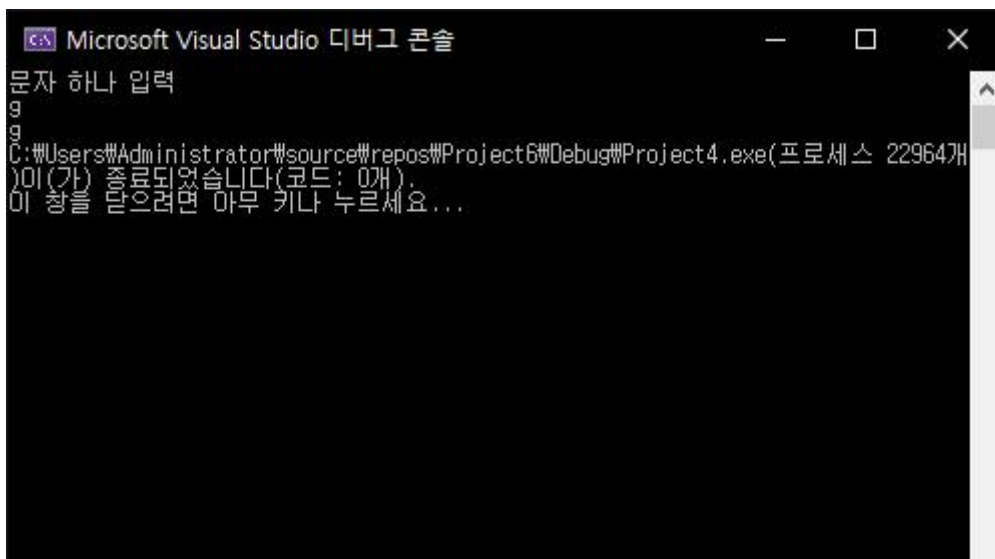
함수 getchar()와 putchar()

함수 getchar()는 영문 'get character'의 의미로 **문자 하나를 입력하는 매크로 함수**이고, putchar()는 'put character'로 반대로 출력하기 위한 매크로 함수이다. 이 함수를 이용하려면 헤더파일 stdio.h 가 필요하다.

getchar() putchar()를 이용한 예제를 만들어보자.

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5
6     puts("문자 하나 입력");
7
8     char ch = getchar(); // getchar() 호출하여 ch변수에 저장
9
10    putchar(ch); // ch에 저장된 값 출력
11
12    return 0;
13 }
```

결과값



```
Microsoft Visual Studio 디버그 콘솔
문자 하나 입력
g
g
C:\Users\Administrator\source\repos\Project6\Debug\Project4.exe (프로세스 22964개)
)이 (가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

scanf_s()의 사용

scanf()는 보안 문제로 더 이상 권장하지 않고 있으며, 사용을하면 경고메세지가 발생한다. 그 대체로 scanf_s()를 권장한다. 하지만 scanf()의 사용권장은 아직 C++ 컴파일러에만 해당 하는 요구사항이다.

scanf()의 경고메세지가 발생하지 않게 하는방법은 아래의 두 문장 중 하나만 선택해서 작성하면 경고메세지는 발생하지 않는다.

```
#define _CRT_SECURE_NO_WARNINGS
#define _CRT_SECURE_NO_DEPRECATED
```

05 연산자

산술연산자

산술연산자는 $+$, $-$, $*$, $/$, $\%$ 로 각각 더하기, 빼기, 곱하기, 나누기, 나머지 연산자이다. 산술연산자의 피연산자는 정수형 또는 실수형이 가능하며, 나머지 연산자는 피연산자로 정수만 가능하다. $\%$ 의 피연산자가 실수이면 오류가 발생한다.

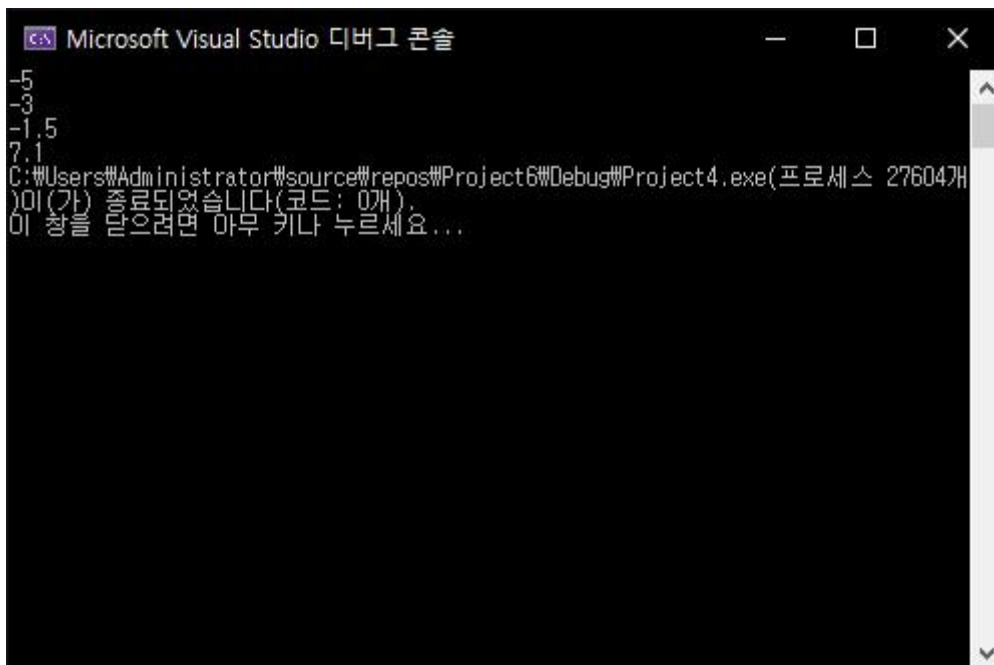
부호연산자

연산자 $+$, $-$ 는 피연산자의 부호를 나타내는 연산자이다. 연산식 $-a$ 는 a 의 부호가 바뀐 값이 결과값이다. 부호연산자는 전위 단항연산자이다.

부호연산자로 예제를 만들어 보자.

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int a = 5;
6      float b = 1.5f;
7
8      printf("%d\n", -a); // a의 부호가 바뀐 값
9      printf("%d\n", -(a - 2)); // a-2의 부호가 바뀐 값
10     printf("%0.1f\n", -b); // b의 부호가 바뀐 값
11     printf("%0.1f", -(-b-5.6)); // -b-5.6의 부호가 바뀐 값
12
13     return 0;
14 }
```

결과값



```
Microsoft Visual Studio 디버그 콘솔
-5
-3
-1.5
7.1
C:\Users\Administrator\source\repos\Project6\Debug\Project4.exe(프로세스 27604개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

대입연산자

대입연산자는 $=$ 으로 오른쪽의 연산값을 변수에 저장하는 연산자이다. 대입연산자의 왼쪽 피연산자는 반드시 하나의 변수만이 올 수 있다.

축약 대입연산자

$a = a + b$ 는 중복된 a 를 생략하고 $a += b$ 로 쓸 수 있다. 이와 같이 산술연산자와 대입연산자를 이어 붙인 연산자 $+=$, $-=$, $/=$, $\%=$ 를 축약 대입연산자라 한다.

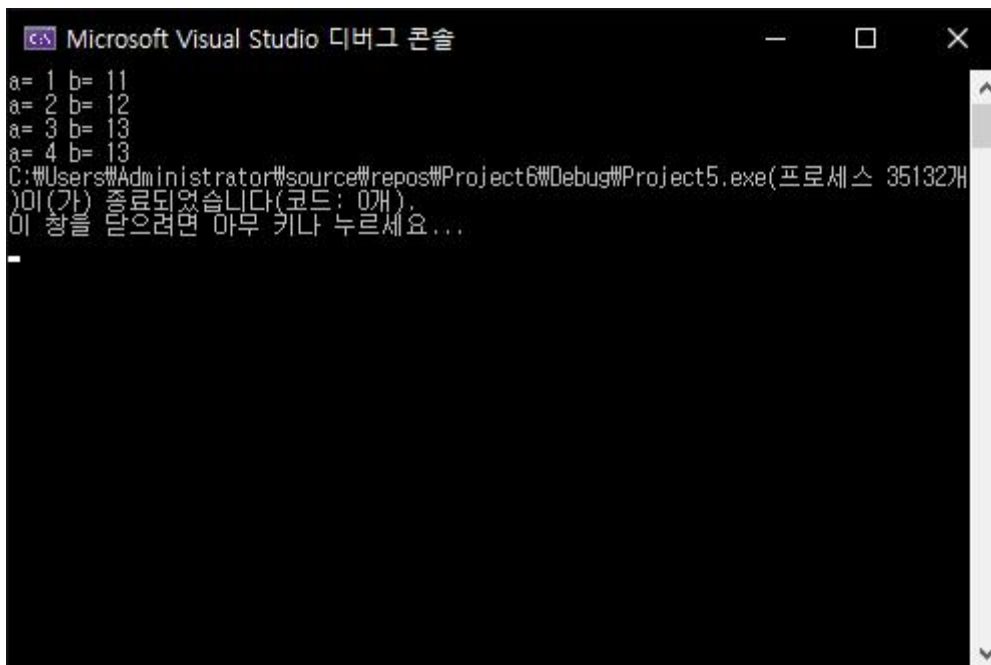
증감연산자

하나의 변수값을 1 증가시키거나 1 감소시키는 문장이다. $++a$, $a++$ 이런식으로 위치를 변경할 수 있는데 위치에 따라 결과값이 달라진다.

결과값은 예제를 통해 알아보자.

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int a = 1;
6     int b = 10;
7
8     printf("a= %d b= %d\n", a++, ++b); // a는 후위증가연산자 b는 전위증가연산자
9     printf("a= %d b= %d\n", a++, ++b);
10    printf("a= %d b= %d\n", a++, ++b); // 세번 실행
11    printf("a= %d b= %d", a, b); // a는 결과값이 달라졌지만 b는 그대로다.
12
13
14 }
```

결과값



```
Microsoft Visual Studio 디버그 콘솔
a= 1 b= 11
a= 2 b= 12
a= 3 b= 13
a= 4 b= 13
C:\Users\Administrator\source\repos\Project6\Debug\Project5.exe(프로세스 35132개)
이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

관계연산자

관계연산자는 두 피연산자의 크기를 비교하기 위한 연산자이다. 연산값은 결과가 참이면 1, 거짓이면 0이다. 관계 연산자의 종류는 $>$, $>=$, $<$, $<=$, $!=$, $==$ 6가지이다.

논리연산자

논리연산자 $\&\&$, $\|\$, $!$ 은 각각 and, or, not의 논리연산을 의미하며, 그 결과가 참이면 1

거짓이면 0을 반환한다. C 언어에서 참과 거짓의 논리형은 따로 없으므로 0,0.0,w0은 거짓을 의미하며, 0이 아닌 경우는 모두 참을 의미한다.

단축평가

논리연산자 &&, ||은 왼쪽 피연산자만으로 논리연산 결과가 결정된다면 오른쪽 피연산자는 평가하지 않는다. 이 방식을 단축평가라고 하며 연산 효율을 높일 수 있다.

&&(and)는 하나라도 거짓이 있으면 거짓이므로 왼쪽 피연산자가 거짓이면 그 결과값은 거짓이고, ||(or)는 하나라도 참이 있으면 참이므로 왼쪽 피연산자가 참이면 그 결과값이 참이다.

연산자 ? :

조건연산자는 조건에 따라 주어진 피연산자가 결과값이 되는 삼항연산자이다. (a ? b : c)에서 a가 참이면 b가 결과값이고 a가 거짓이면 c가 결과값이 된다.

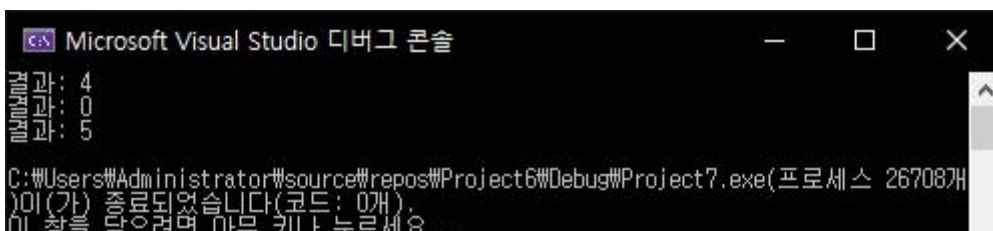
비트 논리 연산자

피연산자 정수값을 비트 단위로 논리 연산을 수행하는 연산자로, &, |, ^, ~ 4가지이다. 피연산자의 자료형은 char, int, long, long long이면 가능하고, 비트 연산은 각 피연산자를 int형으로 변환하여 연산하며 결과도 int형이다.

연산자들로 예제를 한번 만들어보자.

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int a = 5, b = 4, c = 3;
6
7      int result;
8
9      result = a == b ? a : b; // 관계연산자와 조건연산자를 활용해 봤다.
10
11     printf("결과: %d\n", result);
12
13     result = !a && b; // 논리연산자와 단축평가를 활용해 봤다.
14
15     printf("결과: %d\n", result);
16
17     result = a | b; // 비트 논리 연산자를 활용해 봤다.
18
19     printf("결과: %d\n", result);
20
21
22
23     return 0;
24 }
```

결과값



```
Microsoft Visual Studio 디버그 콘솔
결과: 4
결과: 0
결과: 5
C:\Users\Administrator\source\repos\Project6\Debug\Project7.exe(프로세스 26708개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

06 조건과 반복

조건문

if와 switch가 있다.

if는 조건에 따른 선택을 지원하는 구문이다. switch는 주어진 연산식이 문자형 또는 정수형이라면 그 값에 따라 case의 상수값과 일치하는 부분의 문자들을 수행하는 선택구문이다.

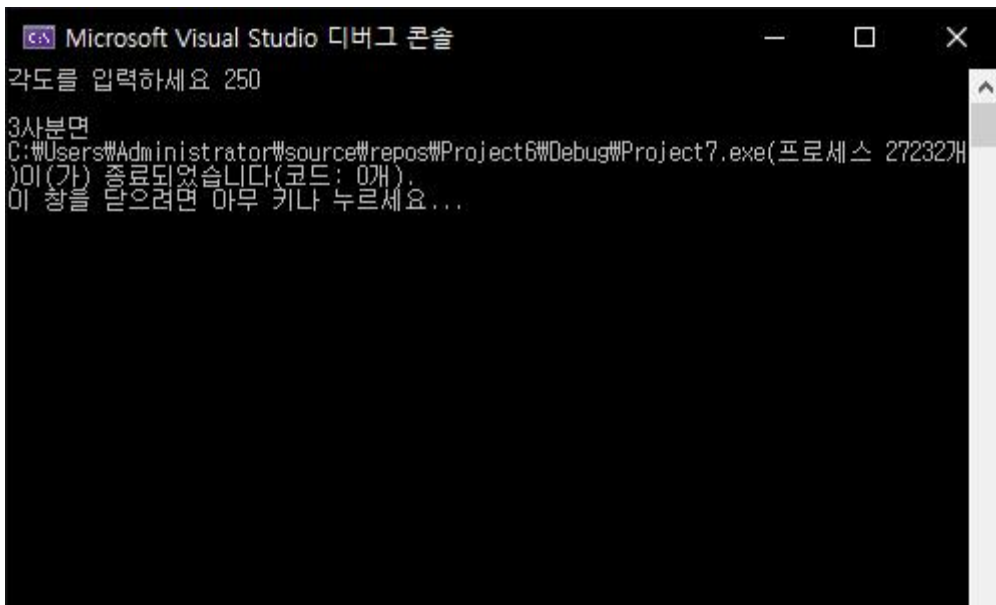
조건문 프로그래밍 연습

입력한 각도에 따라 사분면을 출력하는 프로그램을 작성해봤다.

if문과 switch를 활용해서 작성해봤다.

```
6  main(void)
7
8  int gakdo;
9
10 printf("각도를 입력하세요 ");
11
12 scanf("%d", &gakdo);
13
14 if (0 < gakdo && gakdo < 90) // if문 활용
15 {
16     printf("\n1사분면 ");
17 }
18 else if (90 < gakdo && gakdo < 180) // else if 활용
19 {
20     printf("\n2사분면 ");
21 }
22 else if (180 < gakdo && gakdo < 270)
23 {
24     printf("\n3사분면 ");
25 }
26 else if (270 < gakdo && gakdo < 360)
27 {
28     printf("\n4사분면 ");
29 }
30 else { // else를 활용해 4가지 조건에 만족하지 못하면 switch로 넘어간다
31
32     switch (gakdo)
33     {
34         case 0:
35             printf("양의 X축");
36             break;
37         case 360:
38             printf("양의 X축");
39             break;
40         case 90:
41             printf("양의 Y축");
42             break;
43         case 180:
44             printf("음의 X축");
45             break;
46         case 270:
47             printf("음의 Y축");
48             break;
49     }
50 }
51
52 return 0;
53
54
```

결과값



```
Microsoft Visual Studio 디버그 콘솔
각도를 입력하세요 250
3사분면
C:\Users\Administrator\source\repos\Project6\Debug\Project7.exe(프로세스 27232개)
이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

반복문

반복문은 같거나 비슷한 일을 여러 번 수행하는 구문이다. **while**, **do while**, **for** 세가지 종류의 반복 구문을 제공한다.

while은 조건을 먼저 체크한 뒤, 반복을 실행하고 **do while**은 일단 한번 실행 후 조건을 체크한다. **for**은 숫자로 반복 횟수를 제어하는데 명시적으로 반복 횟수를 결정할 때 주로 사용한다.

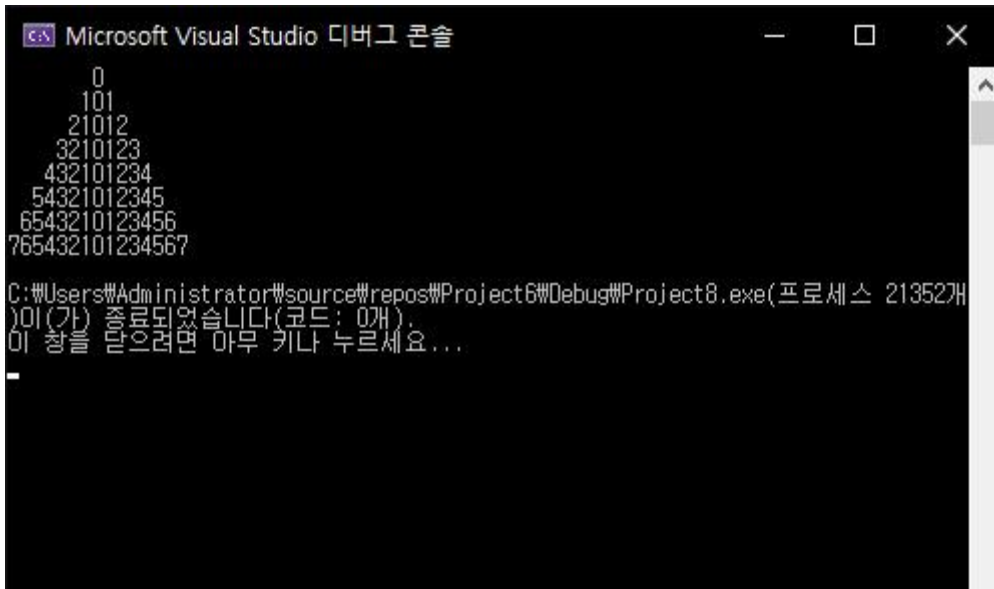
반복문 프로그래밍 연습

중첩 **for**문을 이용해 숫자트리를 출력하는 프로그래밍을 작성해봤다.



```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int num = 0, i = 0, space = 0; //출력할 수, i반복변수, 공백 수
6
7      for (i = 1; i <= 8; i++) // 8줄 생성
8      {
9          for (space = 0; space < (8 - i); space++) // 라인-i번 공백 출력
10         {
11             printf(" ");
12         }
13
14         for (num = (i-1); num+1 > 0; num--) // 숫자가 적어지는 for문
15         {
16             printf("%d", num);
17         }
18
19         for (num = 1; num<i ; num++) // 숫자가 커지는 for문
20         {
21             printf("%d", num);
22         }
23
24         printf("\n");
25     }
```

결과값



```
Microsoft Visual Studio 디버그 콘솔

0
101
21012
3210123
432101234
54321012345
6543210123456
765432101234567

C:\Users\Administrator\source\repos\Project6\Debug\Project8.exe(프로세스 21352개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

08 포인터 기초

포인터 변수와 선언

메모리 주소

메모리 공간은 1바이트(8비트)마다 **고유한 주소**가 있다.

주소연산자&

주소는 변수이름과 같이 저장장소를 참조하는 하나의 방법이다. &가 피연산자인 변수의 메모리 주소를 반환하는 주소연산자이다. &는 변수 앞에서만 쓸 수 있는 전위연산자이고, 상수나 표현식에는 사용 할 수 없다.

포인터 변수

변수의 주소값은 반드시 포인터 변수에 저장해야한다. 일반변수에 저장하면 주소값이라는 의미가 사라진다. 포인터 변수선언에서 자료형과 포인터 변수 이름 사이에 연산자*를 삽입한다. 예를 들어 'int*pt' 이런식으로 선언하며 'int 포인터 pt' 라 읽는다. 어느 변수의 주소값을 저장하려면 반드시 그 변수의 자료유형과 동일한 포인터 변수에 저장해야 한다. 포인터 변수는 참조하는 변수의 종류에 관계없이 모두 4바이트이다.

간접연산자 *

포인터 변수가 갖는 주소로 그 주소의 원래 변수를 참조할 수 있다. 포인터 변수가 가리키고 있는 변수를 참조하려면 간접연산자 *를 사용한다. *를 이용한 참조방식을 예제로 만들어 봤다.

```

1
2 #include <stdio.h>
3
4
5
6 int main (void){
7
8     int data; // data 변수 생성
9     int* pt = &data; // pt 포인터 변수에 data 주소값 참조
10
11     data = 50; // 직접참조로 data의 값을 설정.
12     printf("직접참조로 설정: %d\n", data);
13
14     *pt = 40; // *포인터변수 간접참조로 data의 값을 바꿈.
15     printf("간접참조로 설정: %d", data);
16
17     return 0;
18 }
19
20
21

```

결과값

```

Microsoft Visual Studio 디버그 콘솔
직접참조로 설정: 50
간접참조로 설정: 40
C:\Users\Administrator\source\repos\Project6\Debug\Project9.exe(프로세스 9016개)
이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...

```

주소 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. 더하기와 빼기 연산에는 포인터 변수가 참여할 수 있으나 곱하기와 나누기 연산에는 문법오류가 발생한다.

이웃한 변수의 주소값

int와 int 사이는 주소값으로 그 차이가 절대 값 12 정도이다. int형 자료 크기가 4이므로 나중에 선언된 변수의 주소값을 포인터 변수 p에 저장한다면 이전에 선언된 변수의 주소는 p+3으로 참조할 수 있다.

명시적 형변환

포인터 변수는 동일한 자료형끼리만 대입이 가능하다. 만일 대입문에서 포인터의 자료형이 다르면 경고가 발생한다. 포인터 변수는 자동으로 형변환이 불가능하여 명시적으로 형변환을 해줘야한다.

이중포인터

포인터 변수의 주소값을 갖는 변수를 이중 포인터라고 한다. 이중포인터의 주소값을 갖는 변수는 삼중 포인터라 할 수 있다. 이러한 포인터의 포인터를 모두 다중 포인터라고 한다.

포인터 상수

키워드 `const`를 이용하는 변수 선언은 포인터 변수도 상수로 만들 수 있다. `const` 삽입 위치에 따라 두가지 의미가 있다.

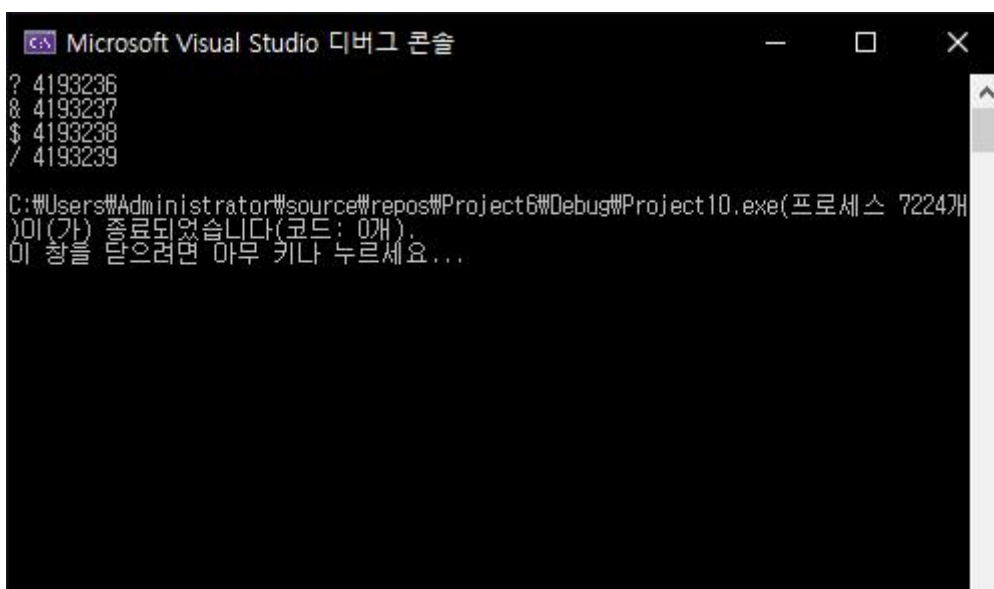
1. <code>const int *pi = &i;</code>	<code>const</code> 가 앞에 나오는 선언은 <code>*pi</code> 로 <code>i</code> 를 수정할 수 없도록 하는 선언 방법이다. 즉 <code>*pi</code> 를 l-value로 사용할 수 없다.
2. <code>int const *pi = &i;</code>	위와 동일한 의미인 사용방법이다.
3. <code>int* const pi = &i;</code>	<code>pi</code> 에 저장되는 초기 주소값을 더 이상 수정할 수 없도록 하는 선언 방법이다. 즉 <code>pi</code> 를 l-value로 사용할 수 없다.

포인터 프로그래밍 연습

`int`형 변수를 형변환하여 `char` 포인터에 저장해 각각의 문자를 출력하는 프로그램을 작성해봤다.

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5
6      int value = 0x2F24263F;
7
8      char* pc = (char*)&value; // value를 형변환하여 *pc에 저장
9
10
11
12     for (int i=0; i <=3; i++)
13     {
14         char ch = *(pc + i); //ch변수에 (pc+i)를 가트키는 문자 변수 저장
15         printf("%c \u %n",ch, pc + i); // 순서대로 문자로 출력
16     }
17
18     return 0;
19 }
```

결과값



```
Microsoft Visual Studio 디버그 콘솔
? 4193236
& 4193237
$ 4193238
/ 4193239

C:\Users\Administrator\source\repos\Project6\Debug\Project10.exe(프로세스 7224개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

09 배열

배열 정의

배열은 동일한 자료 유형이 여러 개 필요한 경우에 유용한 자료 구조이다. 배열은 한 자료유형의 저장공간인 원소를 동일한 크기로 지정된 **배열크기만큼 확보한 연속된 저장공간이다**. 배열에서 중요한 요소는 **배열이름, 원소 자료유형, 배열크기**이다.

배열선언 구문

배열선언 시 배열크기는 양의정수인 상수로 지정하여야 하는데, 리터럴 상수, 매크로 상수 또는 이들의 연산식이 허용된다. 그러나 `const` 상수로는 크기를 지정할 수 없다.

이차원 배열선언

이차원 배열선언은 2개의 대괄호가 필요하다. 첫 번째 대괄호에는 행 크기, 두 번째는 열 크기를 지정한다. 선언 시 초기값을 지정하지 않으면 행과 열 크기는 반드시 명시되어야 한다.

배열과 포인터

```
name[] = {12, 11, 15}; // name이란 이름의 배열
```

배열은 실제 포인터와 연관성이 높는데, 배열의 이름자체가 배열 첫원소의 주소값인 상수이다. 다음과 같은 특징으로 배열이름을 이용하여 모든 배열원소의 주소와 저장값을 참조할 수 있다. 배열이름은 첫 번째 원소의 주소를 나타내는 상수로 `&name[0]`와 같으며, 간접연산자를 이용한 `*name`은 변수 `name[0]`과 같다. `name`으로 연산식도 가능한데 `(name + 1)`이 가능하다 즉 `(name+1)`은 `&name[1]`과 같다.

`*(name+1)`은 `*name[1]`과 같다. 그런데 `(*(name+1))`은 배열의 첫 번째 원소에 1을 더하는 연산식이므로 헷갈리지 않게 주의해야한다.

포인터 변수를 이용한 배열의 원소 참조

참조연산자 `*`의 우선순위는 `++p`의 전위 증감연산자와 같고, 괄호나 `p++`의 후위 증감연산자보다 낮다. 즉 `*p++`은 `*(p++)`과 같다. 연산식이 정리된 표를 보자.

연산식		결과값	연산 후 *p의 값	연산 후 p 증가
<code>++*p</code>	<code>++(*p)</code>	<code>*p + 1</code> : p의 간접참조 값에 1 증가	<code>*p</code> 가 1 증가	p: 없음
<code>*p++</code>	<code>*(p++)</code>	<code>*p</code> : p의 간접참조 값	변동 없음	<code>p+1</code> : p 다음 주소
<code>--*p</code>	<code>--(*p)</code>	<code>*p - 1</code> : p의 간접참조 값에 1 감소	<code>*p</code> 가 1 감소	p: 없음
<code>(*p)--</code>		<code>*p</code> : p의 간접참조 값	<code>*p</code> 가 1 감소	p: 없음

이차원 배열과 포인터

```
td[][3] = {{1,2,3},{4,5,6}}; // td란 이름의 이차원 배열
```

위와 같은 이차원 배열에서 배열이름인 td는 포인터 상수 td[0]을 가리키는 포인터 상수이다. td[0]은 배열의 첫 번째 원소인 td[0][0]의 주소값 &td[0][0]을 갖는 포인터 상수이다. 그러므로 배열이름 td는 이중포인터이다.

포인터 배열과 배열 포인터

포인터 배열

포인터 배열이란 주소값을 저장하는 포인터를 배열 원소로 하는 배열이다.

배열 포인터

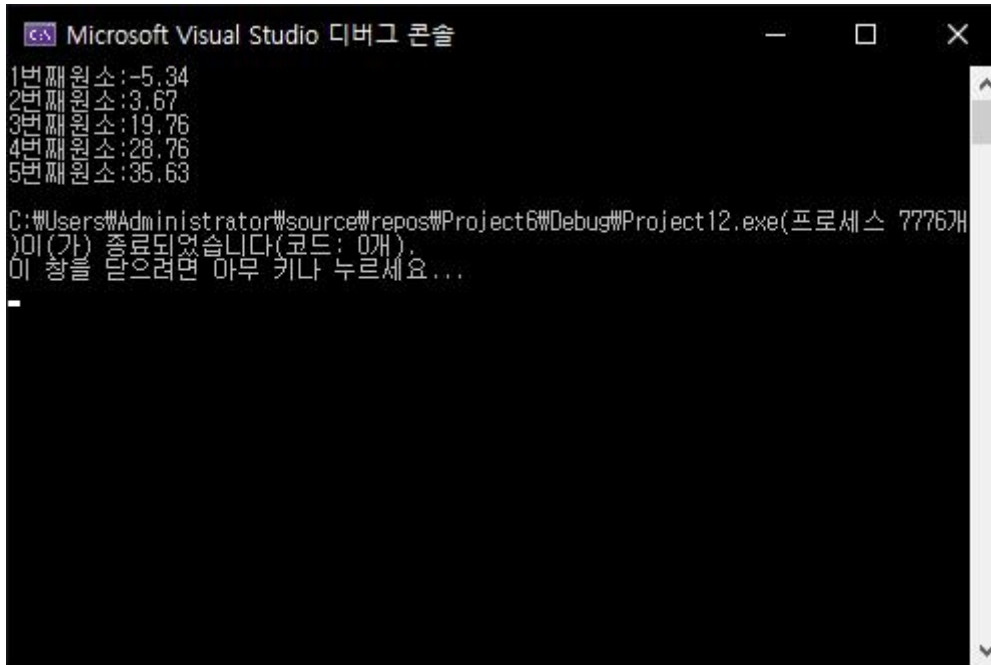
열이 4인 이차원 배열 ary[][4]의 주소를 저장하려면 배열 포인터 변수ptr을 문장 int(*ptr)[4];로 선언해야 한다. 대괄호 사이에 4는 이차원 배열의 열크기이다. 여기서 주의할 점은 괄호(*ptr)이 꼭 필요하다는것인데, 괄호가 없는 int *ptr[4]는 포인터 배열을 선언하는 문장이기 때문이다.

배열 프로그래밍 연습

반복문을 사용하여 배열의 모든 원소를 출력하는 프로그램을 작성해봤다.

```
1  #include <stdio.h>
2
3  int main (void)
4  {
5      double degree[] = { -5.34, 3.67, 19.76, 28.76, 35.63 };
6
7      int size = sizeof(degree) / sizeof(*degree); //배열의 크기 계산
8
9      for (int i = 0; i < size; i++) // 원소를 반복하여 출력
10     {
11         printf("%d번째원소: %0.2f\n", i+1, degree[i]);
12     }
13
14     return 0;
15
16 }
```

결과값



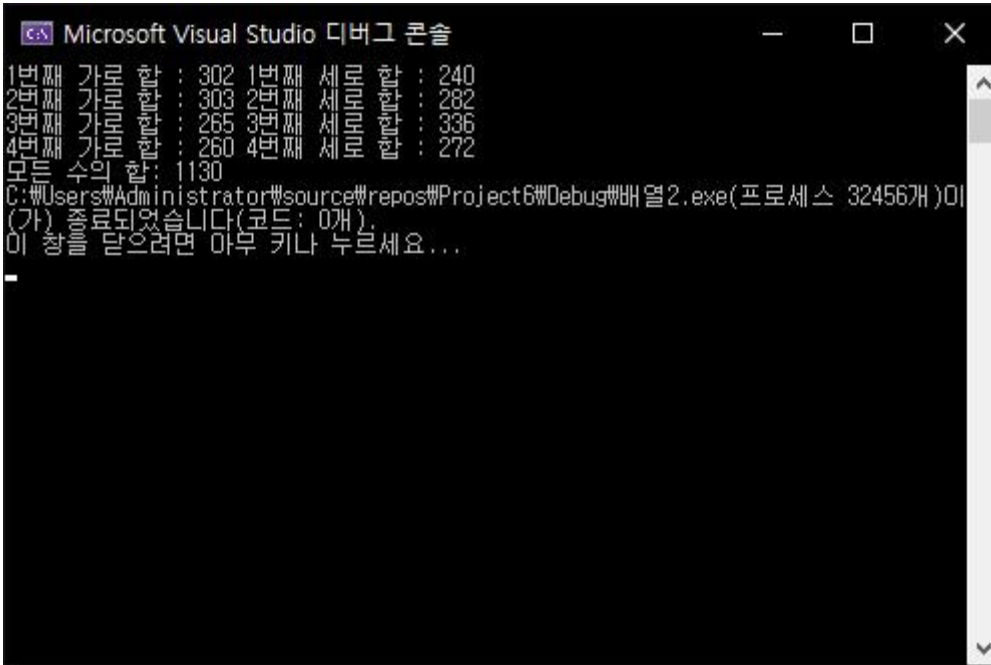
```
Microsoft Visual Studio 디버그 콘솔
1번째 원소:-5.34
2번째 원소:3.67
3번째 원소:19.76
4번째 원소:28.76
5번째 원소:35.63

C:\Users\Administrator\source\repos\Project6\Debug\Project12.exe(프로세스 7776개)
이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

배열의 가로합 세로합 그리고 모든 수의 합을 구하는 프로그램을 작성해봤다.

```
1  #include<stdio.h>
2
3  int main(void)
4  {
5      int ary[4][4] = { {78,48,78,98},{99,92,83,29},
6                      {29,64,83,89},{34,78,92,56} }; // 2차원 배열 생성
7
8      int colsum, rowsum, sum=0;
9
10     for (int i = 0; i < 4; i++)
11     {
12         colsum = 0;
13         rowsum = 0;
14
15         for (int j = 0; j < 4; j++)
16         {
17
18             colsum += ary[i][j]; // 가로합 구하기
19             rowsum += ary[j][i]; // 세로합 구하기
20             sum += ary[i][j]; // 모든 수의 합 구하기
21         }
22         printf("%d번째 가로 합 : %d ", i+1, colsum);
23         printf("%d번째 세로 합 : %d ", i + 1, rowsum);
24         puts("");
25     }
26     printf("모든 수의 합: %d", sum);
27 }
```

결과값



```
Microsoft Visual Studio 디버그 콘솔
1번째 가로 합 : 302 1번째 세로 합 : 240
2번째 가로 합 : 303 2번째 세로 합 : 282
3번째 가로 합 : 265 3번째 세로 합 : 336
4번째 가로 합 : 260 4번째 세로 합 : 272
모든 수의 합: 1130
C:\Users\Administrator\source\repos\Project6\Debug\배열2.exe(프로세스 32456개)이
(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

10 함수 기초

함수개념

함수는 필요한 입력을 받아 원하는 어떤 기능을 수행한 후 **결과를 반환하는 프로그램 단위**. C 프로그램은 최소한 **main()** 함수와 다른 함수로 구성되는 프로그램이다.

라이브러리와 사용자 정의 함수

라이브러리 함수는 이미 개발환경에 포함되어있는 함수를 말하며, 간단히 라이브러리 또는 표준함수라고 부른다. 사용자 정의 함수는 개발자가 직접 개발하는 함수를 말한다. 사용자 정의 함수를 사용하기 위해서는 함수 선언, 함수 호출, 함수 정의가 필요하다.

함수정의

함수정의는 **함수머리**와 **함수몸체**로 구성된다. 함수머리에서는 **반환형,함수이름,매개변수 목록**을 작성하고 함수몸체에서 **프로그램을 구현**한다.

함수실행

정의된 함수를 실행하려면 프로그램 실행 중에 **함수호출**이 필요하다.

함수원형

함수원형은 함수를 선언하는 문장이다. 함수원형은 함수선언으로 변수선언과 같이 함수를 호출하기 전에 반드시 선언되어야 한다.

매개변수

매개변수는 호출하는 부분에서 함수몸체로 값을 전달할 목적으로 이용된다. 매개변수가 필요한 경우 자료형과 변수명의 목록으로 나타내며 필요 없으면 키워드 void를 기술한다.

형식매개변수와 실매개변수

함수정의에서 기술되는 매개변수 목록의 변수를 형식매개변수라 한다. 함수를 호출할 때 기술되는 변수 또는 값을 실매개변수or 실인자or 인자 라고 한다. 형식매개변수는 함수 내부에서만 사용가능한 변수이다.

배열을 매개변수로 사용

배열이름으로 전달

함수의 매개변수로 배열을 전달한다면 한 번에 여러 개의 변수를 전달하는 효과를 가져온다.

이차원 배열이름으로 전달

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

재귀와 함수 구현

재귀 특성

함수구현에서 자신 함수를 호출하는 함수를 재귀함수라고 한다. 재귀적 특성을 표현하는 factorial로 예제를 만들어보자.

```

1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4
5  int factorial(int); // 함수원형 선언
6
7  int main(void)
8  {
9      int num;
10
11     printf("팩토리얼 숫자를 입력하세요: ");
12
13     scanf("%d", &num);
14
15     printf("%d! = %d", num, factorial(num));
16
17     return 0;
18 }
19
20
21 int factorial(int num)
22 {
23     if (num <= 1)
24         return 1; // 1이나 0은 1을 리턴한다.
25     else
26         return (num * factorial(num - 1)); // 1씩 작아지며 반복
27 }

```

☐ (char [29])"팩토리얼 숫자
 온라인 검색

결과값

```

Microsoft Visual Studio 디버그 콘솔
팩토리얼 숫자를 입력하세요: 23
23! = 862453760
C:\Users\Administrator\source\repos\Project6\Debug\팩토리얼.exe(프로세스 9920개)
이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...

```

난수 라이브러리 함수

함수 rand()

특정한 나열 순서나 규칙을 가지지 않는 연속적인 임의의 수를 난수라고 한다. 바로 rand 함수를 이용 해 난수를 생성할 수 있다. 함수 rand()의 함수원형은 **헤더파일 stdlib.h에 정의되어 있다.** 함수 rand()는 0에서 32767사이의 정수 중에서 임의로 하나

의 정수를 반환한다.

함수 srand()

rand()는 호출 순서에 따라 일정한 수가 반환되는 것을 알 수 있다. 이와 다르게 매번 난수를 다르게 생성하려면 시드값을 이용해야한다. 먼저 서로 다른 시드값인 seed를 이용하여 함수 srand(-seed)를 호출한다. 이후 rand()에서 호출하면 서로 다른 난수가 생성된다. 함수 **time(NULL)**은 **1970년 1월1일 이후 현재까지 경과된 시간을 초 단위로 반환하는 함수**이다. 여기서 NULL은 포인터로서 정수의 0이나 실수의 0.0과 같이 0번지의 주소값을 나타내는 상수이다. time() 이용하려면 헤더파일 time.h를 삽입해야 한다. 난수에 시드를 지정하기 위해 **srand((long) time(NULL))**을 호출한다. 이 이후부터 rand()로 난수를 생성하면 서로 다른 난수를 만들 수 있다.

난수발생 프로그래밍 순서

time() 이용하기 위해 time.h 삽입



난수에 시드하기 위해 srand((long) time(NULL))호출



1에서 n까지 난수 생성 위해 rand() % n+ 1을 이용



일반화 시켜 **a~b에서 난수를 발생 시키려면** 수식 **rand() % (b - a +1) +a**를 이용

수학과 문자 라이브러리 함수

math.h

수학 관련 함수를 사용하려면 헤더파일 math.h을 삽입해야한다.

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
double exp(double x)	ex
double log(double x)	loge(x)
double log10(double x)	log10(x)
double pow(double x, double y)	xy
double ceil(double x)	x보다 작지 않은 가장 작은 정수 (올림)
double floor(double x)	x보다 크지 않은 가장 큰 정수 (내림)
int abs(int x)	정수 x의 절대 값
double fabs(double x)	실수 x의 절대 값

ctype.h

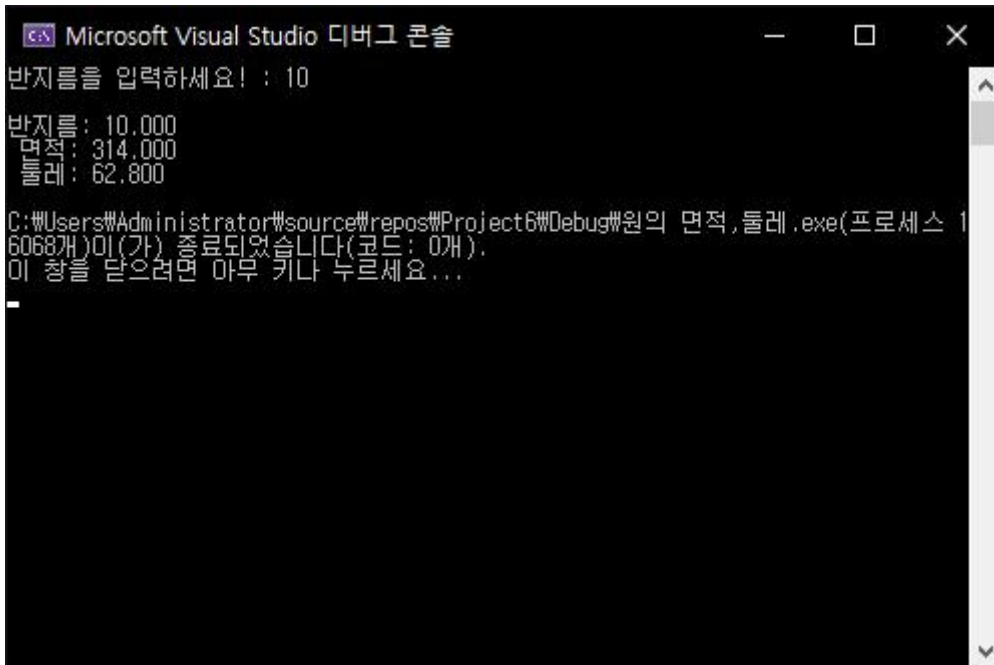
문자 관련 함수를 사용하려면 헤더파일 ctype.h를 삽입해야한다. 문자 관련 함수는 문자를 검사하거나 변환하는 기능을 수행하는데 일반적으로 검사는 isxxxx(char) 으로 변환은 toxxx(char)로 명명된다. 검사함수는 0(false)과 0이 아닌 정수값(true)을 반환하며, 변환 함수는 변환된 문자를 반환한다.

프로그래밍 연습

반지름을 입력 받아 원의 면적과 둘레의 길이를 구하는 프로그램을 작성해봤다.

```
2  #define _CRT_SECURE_NO_WARNINGS
3
4  #include<stdio.h>
5  #define PI 3.14 // 파이 매크로 상수 지정
6
7  double area(double r); // 면적구하기 함수 원형
8
9  double circumference(double r); // 둘레구하기 함수 원형
10
11 int main(void)
12 {
13     double num;
14     printf("반지름을 입력하세요! : ");
15     scanf("%lf", &num); //double 인자 받을 시 %f 사용하면 오류 발생
16
17     puts("");
18     printf("반지름: %0.3f\n ", num); // 반지름
19     printf("면적: %0.3f\n ", area(num)); //area함수로 면적 출력
20     printf("둘레: %0.3f\n ", circumference(num)); // circum함수로 둘레 출력
21
22     return 0;
23 }
24
25 double area(double r) // 면적구하기 함수 정의
26 {
27     double areareult;
28
29     areareult = PI * r * r; // 면적 구하기 식
30
31
32     return areareult; // 면적 리턴
33 }
34
35 double circumference(double r) // 둘레구하기 함수 정의
36 {
37     double circumreault;
38
39     circumreault = 2 * PI * r; // 둘레 구하기 식
40
41     return circumreault; // 둘레 리턴
42 }
43
```

결과값



```
Microsoft Visual Studio 디버그 콘솔
반지름을 입력하세요! : 10
반지름: 10.000
면적: 314.000
둘레: 62.800
C:\Users\Administrator\source\repos\Project6\Debug\원의_면적_둘레.exe(프로세스 16068)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

11 문자와 문자열

문자와 문자열의 개념

문자는 영어의 알파벳이나 한글의 한 글자를 작은 따옴표로 둘러싸서 'A'와 같이 표기하며, 자료형 char로 지원한다. 작은 따옴표에 의해 표기된 문자를 문자 상수라 한다. 문자의 모임인 일련의 문자를 문자열이라한다. 문자열은 일련의 문자 앞 뒤로 큰 따옴표로 둘러싸서 "JAVA"로 표기한다. 이처럼 큰 따옴표에 의해 표기된 문자를 문자열 상수라 한다. 문자의 나열인 문자열은 'ABC'처럼 작은 따옴표로 둘러싸도 문자가 될 수 없으며, 오류가 발생한다.

문자와 문자열의 선언

C에서는 char형 변수에 문자를 저장한다. 하지만 문자열을 저장하기 위한 자료형이 따로 없는데, 문자열을 저장하려면 '문자 배열'을 사용한다. 하지만 여기에는 주의점이 있다. 바로 문자열의 마지막을 의미하는 NULL 문자 '\0'이 마지막에 저장되어야 한다. 그러므로 문자열이 저장되는 배열크기는 반드시 저장될 문자 수 +1 이여야 한다.

문자와 문자열 출력

printf()에서 %c로 문자를 출력할 수 있고, 배열이름, 또는 문자 포인터를 사용하여 %s로 문자열을 출력할 수 있다.

문자 포인터

문자열을 처리하는 다른 방법은 문자열 상수를 문자 포인터에 저장하는 방식이다. 하지만 문자 포인터에 의한 선언으로는 문자 하나 하나의 수정은 할 수 없다.

'\0' 문자에 의한 문자열 분리

printf()에서 %s는 문자 포인터가 가리키는 위치에서 NULL 문자까지를 하나의 문자열로 인식한다. 그러므로 같은 문자 배열에 있더라도 char c[]="Hello\0World!"; 이라면 printf에서 %s로 c를 출력하면 Hello만 나오게 된다.

다양한 문자 입출력

문자입력 함수가 정리된 표를 보자.

함수	scanf("%c",&ch)	getchar()	getche() _getche()	getch() _getch()
헤더 파일	stdio.h		conio.h	
버퍼 이용	버퍼 이용함		버퍼 이용 안함	
반응	엔터키를 눌러야 작동		문자 입력마다 반응	
입력 문자의 표시	누르면 바로 표시		누르면 바로 표시	표시 안됨
입력문자 수정	가능		불가능	

다양한 문자열 라이브러리 함수

문자의 배열 관련 함수는 string.h에 함수원형이 정의되어있다.

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

문자열 프로그래밍 연습

문자열을 표준입력받아 구두점의 수를 구하여 출력하는 프로그램을 작성해봤다.

```

1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3  #include <ctype.h>
4
5  int main() {
6      char str[101];
7
8      printf("입력하세요.\n");
9
10     gets(str);
11
12
13     int i = 0;
14     int cnt = 0;
15
16     while (str[i] != '\0')
17     {
18         if (ispunct(str[i]))
19             cnt++;
20
21         i++;
22     }
23
24     printf("구두점의 수 : %d ", cnt);
25
26
27 }
28
29
30

```

결과값

```

Microsoft Visual Studio 디버그 콘솔
입력하세요.
hi, my name is hyenbok, i like kimch.
구두점의 수 : 3
C:\Users\Administrator\source\repos\과제\Debug\과제.exe(프로세스 40948개)이(가)
종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버
깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...

```

12 변수 유효범위

전역변수와 지역변수

변수 scope

변수의 참조가 유효한 범위를 변수의 유효 범위(scope)라 한다. 유효 범위는 크게 지역 유효 범위와 전역 유효 범위로 나눌 수 있다. 지역 유효 범위는 함수 또는 블록 내에서 선언되어 그 지역에서 변수의 참조가 가능한 범위이다. 전역 유효 범위는 다시 2가지로 나눌 수 있다. 하나는 하나의 파일에서만 변수의 참조가 가능한 범위고, 하나는 프로젝트를 구성하는 모든 파일에서 변수의 참조가 가능한 범위이다.

지역변수

카드를 예로 들어보자. **국내 전용 카드가 지역변수라면 국내외 사용카드는 전역 변수**라 할 수 있다. 지역변수는 함수 또는 블록에서 선언된 변수이다. 내부변수 또는 자동변수 라고도 부른다.

1. 함수나 블록에서 지역변수는 선언 문장 이후에 함수나 블록의 **내부에서만 사용이 가능하다**. 다른 함수나 블록에서는 사용이 불가능하다.
2. 함수의 매개변수도 함수 전체에서 사용 가능한 지역변수와 같다.
3. 지역변수는 선언 후 초기화 하지 않으면 쓰레기 값이 저장되므로 주의 해야한다.
4. 지역변수가 할당되는 메모리 영역을 스택이라고 한다. 지역변수는 선언된 부분에서 자동 생성되고 함수나 블록이 종료되는 순간 메모리에서 자동 삭제된다. 이러한 이유 때문에 자동변수라 하기도 한다.

전역변수

전역변수는 함수 외부에서 선언되는 변수이다. 전역변수는 외부변수라고도 하며, 전역변수는 일반적으로 프로젝트의 모든 함수나 블록에서 참조할 수 있다.

1. 전역변수는 선언되면 자동으로 초기값이 자료형에 맞는 0이 저장이 된다.
2. 함수나 블록 안에서 전역변수와 같은 이름으로 지역변수를 선언할 수 있다. 이럴 경우, 함수 내부나 블록에서는 그 지역변수만 사용할 수 있고 전역변수는 인식하지 못한다.
3. 전역변수는 프로젝트의 다른 파일에서도 참조가 가능하다. 다른파일에서 사용하려면 키워드 `extern`을 사용하여야 한다. `extern`은 변수선언 맨 앞에 넣어야 한다. `extern`을 사용할 때 자료형은 생략가능하다. `extern`을 사용한 변수 선언은 새로운 변수 선언이 아닌, 단지 이미 존재한 전역변수의 유효 범위를 확장 시키는 것이다.

기억 부류와 레지스터변수

변수 선언의 위치에 따라 변수는 전역과 지역으로 나뉜다. 마찬가지로 변수는 4가지의 기억부류인 **auto, register, static, extern**에 따라 할당되는 **메모리 영역이 결정되고 메모리 할당과 제거 시기가 결정된다**.

전역과 지역은 선언 위치에 따라 구분되나 기억 부류는 키워드에 의해 구분되므로 훨씬 구분하기는 쉽다. 아래 의 표를 참고해 기억부류 키워드의 적용 범위를 알아보자.

기억부류 종류	전역	지역
auto	x	o
register	x	o
static	o	o
extern	o	x

auto, register, static은 새로운 변수의 선언에 사용되는 키워드이나, **extern**은 이미 존재하는 함수를 이제 사용하겠다는 것을 알리는 구문에 사용되는 키워드이다. 다만

extern이 선언되는 위치에 따라 이변수의 사용 범위는 전역 또는 지역으로 한정 될 수 있다. 기억부류 구문은 변수 선언 문장에서 자료형 앞에 하나의 키워드를 넣는 방식이다. **extern을 제외한 나머지 3개의 기억부류의 변수선언에서 초기값을 지정할 수 있다.** auto는 지역변수 선언에서 사용되며 생략 가능하다. 즉 지금까지 함수에 선언된 모든 변수가 auto가 생략된 자동변수였다.

키워드 register

레지스터 변수는 변수의 저장공간이 일반 메모리가 아니라 레지스터에 할당되는 변수이다.

1. 레지스터 변수는 키워드 register를 자료형 앞에 넣어 선언한다.
2. 지역변수에서만 사용이 가능하며, 함수나 블록이 시작하면 레지스터에 값이 저장되고, 함수나 블록을 빠져나오면서 소멸되는 특성을 갖는다.
3. 일반 메모리보다 빠르게 참조할 수 있어, 레지스터 변수는 처리속도가 더 빠르다.
4. 일반 메모리에 할당되는 변수가 아니기 때문에 **주소연산자&를 사용할 수 없다.**

이러한 특성으로 주로 레지스터 변수는 처리 속도를 증가시키려는 변수에 이용한다. 특히 반복문의 횟수를 제어하는 제어변수에 이용하면 효과적이다.

키워드 static

변수 선언에서 static을 넣으면 정적변수를 선언할 수 있다. 정적변수는 **정적 지역변수**와 **정적 전역변수**로 나눌 수 있다.

1. 정적변수는 생성된 이후 메모리에서 제거되지 않으므로 지속적으로 저장값을 유지하거나 수정 할 수 있는 특성이 있다.
2. 정적변수는 프로그램이 시작되면 메모리에 할당되고, 프로그램이 종료되면 메모리에서 제거된다.
3. 정적변수는 초기값을 지정하지 않으면 자동으로 자료형에 맞는 0값이 저장된다.
4. 정적변수의 초기화는 단 한번만 수행된다. 그러므로 실행 중간에 더 이상 초기화 되지 않는 특성이 있으며, 초기화는 상수로만 가능하다.

정적 지역변수

함수나 블록에서 정적으로 선언되는 변수가 정적 지역변수이다. 정적 지역변수는 함수나 블록을 종료해도 메모리에서 제거되지 않고 계속 메모리에 유지 관리되는 특성이 있다.

정적 전역변수

함수 외부에서 정적으로 선언되는 변수가 정적 전역변수이다. 정적 전역변수는 선언된 파일 내부에서만 참조가 가능하다. 즉 정적 전역변수는 extern에 의해 다른 파일에서 참조가 불가능하다. 프로그램이 크고 복잡하면 전역변수의 사용은 원하지 않는 전

역변수의 수정과 같은 부작용의 위험성이 항상 존재한다. 그러므로 필요할 때가 아니라면 정적 전역변수를 이용하는 것이 바람직하다.

메모리 영역

메인 메모리 영역은 프로그램 실행 과정에서 데이터영역, 힙영역, 스택영역, 세 부분으로 나뉜다. 메모리 영역은 변수의 유효범위와 생존기간에 결정적 역할을 하며, 변수는 기억부류에 따라 할당되는 메모리 공간이 달라진다.

1. 기억부류는 변수의 유효범위와 생존기간을 결정한다.
2. 기억부류는 변수의 저장공간의 위치가 데이터, 힙, 스택 영역인지도 결정하며, 초기값도 결정한다.
3. 데이터 영역은 전역변수와 정적변수가 할당되는 공간이다.
4. **힙 영역은 동적할당되는 변수가 할당되는 저장공간이다.**
5. 스택 영역은 함수 호출에 의한 형식 매개변수 그리고 함수 내부의 지역변수가 할당되는 저장공간이다.
6. 데이터 영역은 주소가 낮은 값에서 높은 값으로 저장 장소가 할당된다. 프로그램이 시작된 시점에서 정해진 크기대로 고정된 메모리 영역이 확보되는데, 힙 영역과 스택 영역은 영역 크기가 계속적으로 변한다.
7. 스택 영역은 주소가 높은 값에서 낮은 값으로 저장 장소가 할당된다. 함수 호출과 종료에 따라 지속적으로 할당되었다가 제거되는 작업을 반복한다.
8. 힙 영역은 데이터 영역 - 스택 영역 사이에 위치한다. 힙 영역은 낮은 값에서 높은 값으로 사용하지 않는 공간이 동적으로 할당된다.

변수의 이용기준

1. 함수나 블록이 종료되더라도 지속적으로 값을 저장하고 싶을 때는 정적 지역변수를 사용한다.
2. 해당 파일 내부에서만 변수를 공유하고자 하는 경우는 정적 전역변수를 이용한다.
3. 프로그램에서 모든 영역에서 값을 공유하고자 하면 전역변수를 이용하지만, 가급적으로 사용을 줄이는 것이 프로그램 문제를 줄일 수 있다.

13 구조체와 공용체

구조체 개념과 정의

구조체 개념

정수나, 문자, 실수나 포인터 그리고 이들의 배열 등을 묶어 하나의 자료형으로 이용하는 것이 구조체이다. 연관성이 있는 서로 다른 개별적인 자료형의 변수들을 하나의 단위로 묶은 새로운 자료형을 구조체라한다. 구조체는 연관된 멤버로 구성되는 통합 자료형으로 대표적인 유도 자료형이다. 즉 기존 자료형으로 새로이 만들어진 자료형을

유도 자료형이라고 한다.

구조체 정의

구조체를 자료형으로 사용하려면 먼저 구조체를 정의해야한다. 구조체를 사용하려면 먼저 구조체를 만들 구조체 틀을 정의해야한다. 구조체를 정의하는 방법은 키워드 **struct** 다음에 구조체 태그이름을 기술하고 중괄호를 이용하여 원하는 멤버를 여러 개의 변수로 선언하는 구조이다. 구조체를 구성하는 하나 하나의 항목은 구조체 멤버 또는 필드라 한다. 구조체 정의는 변수의 선언과는 다른 것으로 변수 선언에서 이용될 새로운 구조체 자료형을 정의하는 구문이다. 한 구조체 내부에서 선언되는 구조체 멤버의 이름은 모두 유일해야 한다. 구조체 멤버로는 일반 변수, 포인터 변수, 배열, 다른 구조체 변수 및 구조체 포인터도 허용된다.

구조체 변수 선언

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

구조체 변수 초기화

초기화 값은 다음과 같이 중괄호 내부에서 구조체의 각 멤버 정의 순서대로 초기값을 쉼표로 구분하여 기술한다. 배열과 같이 초기값에 기술되지 않는 멤버값은 자료형에 맞는 0 으로 저장된다.

```
struct 구조체이름 변수이름 = {초기값1, 초기값2, 초기값3};
```

구조체의 멤버 접근 연산자 .와 변수 크기

선언된 구조체형 변수는 접근연산자 `.`를 사용하여 멤버를 참조할 수 있다. 즉 문장 `yours.actnum= 1002;`은 변수 `yours`의 멤버 `actnum`에 1002를 저장하는 기능을 수행한다. 이와 같이 `yours.name`과 `yours.balance`로 구조체 멤버 `name`과 `balance`도 참조할 수 있다. 접근연산자 `.`은 참조연산자라고도 부른다. 실제 구조체의 크기는 멤버의 크기의 합보다 크거나 같다.

구조체 멤버로 사용되는 구조체

구조체 멤버로 이미 정의된 다른 구조체 형 변수와 자기 자신을 포함한 구조체 포인터 변수를 사용 할 수 있다. 구조체 멤버로 다른 구조체를 허용하는 예제를 만들어 봤다.

```

1  #include <stdio.h>
2  #include <string.h>
3
4  struct human // 사람의 키, 몸무게, 나이를 알기 위한 구조체
5  {
6      int height; // 키
7      int weight; // 몸무게
8      int age; // 나이
9      char name[20]; // 이름
10 }
11
12
13 struct school // 학교 정보를 담은 구조체
14 {
15     struct human student; // 학생정보
16     char name[20]; // 학교이름
17     int number; // 전화번호
18 };
19
20
21 int main(void)
22 {
23     struct school me = { { 158, 50, 17, "심언어"}, "짱짱고등학교", 123456 };
24
25     printf("구조체 크기: %d\n", sizeof(me));
26
27     printf("[이름: %s]\n", me.student.name); // 중첩 구조체는 접근 연산자를 2번 사용
28     printf("[나이: %d]\n", me.student.age);
29     printf("[키 : %d]\n", me.student.height);
30     printf("[몸무게: %d]", me.student.weight);
31 }

```

결과값

```

Microsoft Visual Studio 디버그 콘솔
구조체 크기: 56
[이름: 심언어]
[나이: 17]
[키 : 158]
[몸무게: 50]
C:\Users\Administrator\source\repos\Project6\Debug\구조체.exe(프로세스 28704개)
이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...

```

구조체 변수의 대입과 동등비교

동일한 구조체형의 변수는 대입문이 가능하다. 같은 구조체형의 변수 a와 b에서 동등비교는 할 수 없다. 그러므로 구조체를 비교한다면 구조체 멤버, 하나 하나를 비교해야 한다. 문자열 멤버는 라이브러리 strcmp()를 사용하면 a와 b 내용을 모두 비교할 수 있다.

공용체 개념

동일한 저장 장소에 여러 자료형을 저장하는 방법으로, 공용체를 구성하는 멤버에 한번에 한 종류만 저장하고 참조할 수 있다.

공용체(union)는 서로 다른 자료형의 값을 동일한 저장공간에 저장하는 자료형이다. 공용체 변수의 크기는 멤버 중 가장 큰 자료형의 크기로 정해진다.

1. 공용체의 멤버는 모든 멤버가 동일한 저장 공간을 사용하므로, 동시에 저장하여 이용할 수 없고, 마지막으로 저장된 하나의 멤버 자료값만을 저장한다.
2. 공용체도 구조체와 같이 typedef를 이용하여 새로운 자료형으로 정의 가능하다.
3. 공용체의 초기화 값은 처음 선언된 멤버의 초기값으로만 저장된다. **공용체 변수로 멤버를 접근하기 위해서는 구조체와 같이 접근연산자 . 를 사용한다.**

typedef 구문

typedef는 이미 사용되는 자료 유형을 다른 새로운 자료형 이름으로 재정의할 수 있도록 하는 키워드이다. 자료형을 재정의하는 이유는 프로그램의 시스템 간 호환성과 편의성을 위해 필요하다. **typedef도 일반 변수와 같이 그 사용 범위를 제한한다.** 함수 내부에서 재정의 되면 선언된 이후의 그 함수에서만 이용이 가능하다.

구조체 자료형 재정의

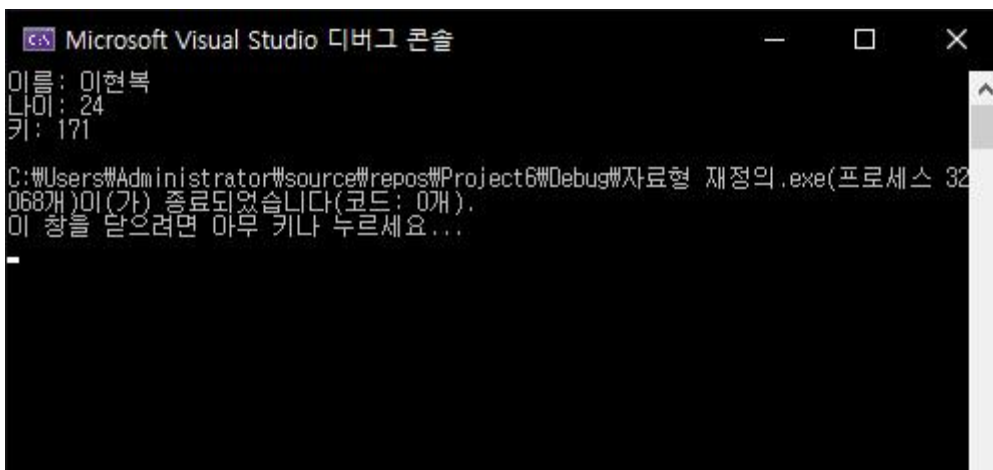
구조체 struct date가 정의된 상태에서 **typedef 사용하여 구조체 struct date를 date로 재정의** 할 수 있다. typedef를 구조체 정의 할 때 함께 처리하면 더욱 쉽게 처리 할 수 있다. 구조체 자료형을 재정의 하는 예제를 만들어봤다.


```

1  #include <stdio.h>
2
3  typedef char ch; // char 재정의
4  typedef int smile; // int 재정의
5
6  typedef struct //구조체 정의와 동시에 자료형 human으로 재정의
7  {
8      ch name[20]; // 이름
9      smile age; // 나이
10     smile height; // 키
11
12 }human;
13
14
15 smile main(void)
16 {
17     human hyeonbok = { "이현복", 24, 171 }; // human 자료형에 hyeonbok 초기화
18
19     printf("이름: %s\n", hyeonbok.name); // 이름 출력
20     printf("나이: %d\n", hyeonbok.age); // 나이 출력
21     printf("키: %d\n", hyeonbok.height); // 키 출력
22
23     return 0;
24 }
25
26
27
28

```

결과값



```

Microsoft Visual Studio 디버그 콘솔
이름: 이현복
나이: 24
키: 171
C:\Users\Administrator\source\repos\Project6\Debug\자료형 재정의.exe(프로세스 32068개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...

```

구조체 포인터

포인터는 각각의 자료형 저장 공간의 주소를 저장하듯이 구조체 포인터는 구조체의 주소값을 저장할 수 있는 변수이다. 구조체 포인터 변수는 구조체이름*변수이름 으로 선언된다. 구조체 포인터는 다른 포인터와 사용 방법이 동일하다.

구조체 멤버 접근 연산자 ->

멤버 접근 연산자 ->는 p->name과 같이 사용한다. p->name은 포인터 p가 가리키는 구조체 변수의 멤버 name을 접근하는 연산식이다.

1. 접근연산자인->는 두 문자가 연결된 하나의 연산자 이므로 -와> 사이에 공백이 들어가서는 안된다.

2. p->은 연산식 (*p).name으로도 사용 가능하다. 그러나 (*p).name은 *p.name과 다르다.

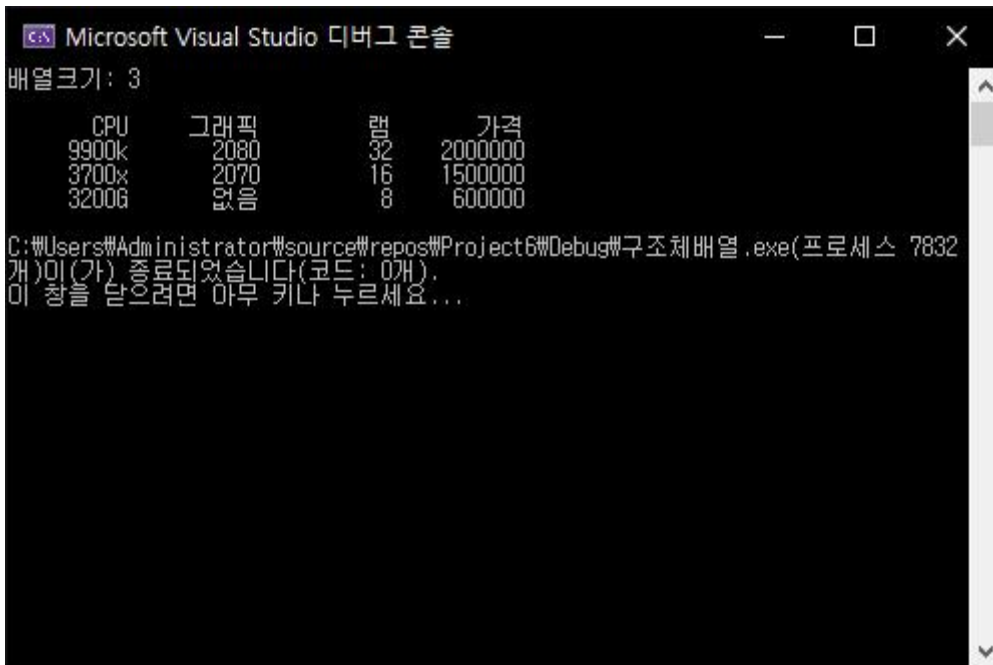
3. 연산식 *p.name은 접근연산자(.)가 간접연산자(*)보다 우선순위가 빠르므로 *(p.name)과 같은 연산식이다. p는 포인터 이므로 문법오류가 발생한다.

구조체 배열 변수 선언

다른 배열과 같이 동일한 구조체 변수가 여러 개 필요하면 구조체 배열을 선언하여 이용할 수 있다. 구조체 배열을 선언하는 예제를 만들어 봤다.

```
2  #include <stdio.h>
3
4  typedef struct computer // 선언과 동시에 재정의
5  {
6      char cpu[20];        //cpu 모델명
7      char graphics[20];   // 그래픽카드 모델명
8      int ram;             // 램 n 기가
9      int price;          // 가격
10 }computer;
11
12
13 char* head[] = { "CPU", "그래픽", "램", "가격" };
14
15 int main(void)
16 {
17     // 구조체 배열 선언 및 초기화
18     computer mycom[] = { { "9900k", "2080", 32, 2000000 }
19                          , { "3700x", "2070", 16, 1500000 }
20                          , { "3200G", "없음", 8, 600000 } };
21
22     int arysize = sizeof(mycom) / sizeof(mycom[0]); // 배열크기 구하기
23
24     printf("배열크기: %d\n\n", arysize);
25
26     printf("%10s %10s %10s %10s\n", head[0], head[1], head[2], head[3]);
27
28     for (int i = 0; i < arysize; i++) // 구조체의 모든 원소 순서대로 출력
29     {
30         printf("%10s %10s %10d %10d\n", mycom[i].cpu, mycom[i].graphics,
31             mycom[i].ram, mycom[i].price);
32     }
33     return 0;
34 }
```

결과값



```
Microsoft Visual Studio 디버그 콘솔
배열크기: 3
    CPU    그래픽    램    가격
9900k    2080    32    2000000
3700x    2070    16    1500000
3200G    없음    8    600000
C:\Users\Administrator\source\repos\Project6\Debug\구조체배열.exe(프로세스 7832
개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

머리말 포트폴리오를 마치며...

포트폴리오 작성을 마치며... 책을 작성하는데 수 많은 노력과 시간이 필요하다 느끼고 길고 힘들었던 과제를 마치는거 같아 굉장히 개운합니다. 퀄리티가 좋다고 자부할 수는 없지만 책을 되짚어 보며 모르고 넘어갔던 내용... 중요하다 생각되는 내용... 위주로 담아봤습니다. 감사합니다.