

C Programming

Ch.2-2 상수와 기본 자료형

Contents

1. C 언어가 제공하는 기본 자료형의 이해
2. 문자의 표현방식과 문자를 위한 자료형
3. 상수에 대한 이해
4. 자료형의 변환

자료형은 데이터를 표현하는 방법

1. 실수를 저장할 것이냐? 정수를 저장할 것이냐!
 - 값을 저장하는 방식이 실수냐 정수냐에 따라서 달라짐
2. 얼마나 큰 수를 저장할 것이냐!
 - 큰 수를 표현하기 위해서는 많은 수의 바이트 필요

이름 이외에 메모리 공간의 할당을 위해 필요한 두 가지 정보

↓ 요청의 예

"아! 제가 정수를 저장할건데요. 크기는 4바이트로 하려고 합니다. 그 정도면 충분할거예요. 그리고 변수의 이름은 *num*으로 할게요."

↓ C언어에서의 예

```
int num;
```

자료형 : 데이터 표현 방법 → C 언어가 제공하는 기본 자료형들은?

기본 자료형의 종류와 데이터의 표현 범위

- ▶ 정수형과 실수형으로 구분
 - 표현 범위에 따라 각각 여러 개의 자료형으로 나뉨
 - 실제 바이트 크기 및 표현 범위는 컴파일러에 따라 달라질 수 있음
 - 예) int : 2바이트 또는 4바이트

	자료형	크기	값의 표현범위
정수형	char	1바이트	-128이상 +127이하
	short	2바이트	-32,768이상 +32,767이하
	int	4바이트	-2,147,483,648이상 +2,147,483,647이하
	long	4바이트	-2,147,483,648이상 +2,147,483,647이하
	long long	8바이트	-9,223,372,036,854,775,808이상 +9,223,372,036,854,775,807이하
실수형	float	4바이트	$\pm 3.4 \times 10^{-37}$ 이상 $\pm 3.4 \times 10^{+38}$ 이하
	double	8바이트	$\pm 1.7 \times 10^{-307}$ 이상 $\pm 1.7 \times 10^{+308}$ 이하
	long double	8바이트 이상	double 이상의 표현범위

sizeof 연산자를 이용한 바이트 크기 확인

- ▶ sizeof(변수 또는 상수 또는 자료형)
 - sizeof(int) : int 타입의 메모리 크기(바이트 수) 반환
 - 피연산자로 자료형이 올 경우 괄호 사용, 그 외는 옵션

```
int main(void)
{
    char c = 10;
    int i = 3048;
    double f = 3.1415;

    printf("변수 c의 크기 : %d \n", sizeof c);
    printf("변수 i의 크기 : %d \n", sizeof(i));
    printf("변수 f의 크기 : %d \n", sizeof f);

    printf("char 할당 메모리 : %d \n", sizeof(char));
    printf("int 할당 메모리 : %d \n", sizeof(int));
    printf("double 할당 메모리: %d \n", sizeof(double));

    printf("float 할당 메모리 : %d \n", sizeof(float));
    printf("double 할당 메모리: %d \n", sizeof(double));
}
```

변수 c의 크기	: 1
변수 i의 크기	: 4
변수 f의 크기	: 8
char 할당 메모리	: 1
int 할당 메모리	: 4
double 할당 메모리	: 8
float 할당 메모리	: 4
double 할당 메모리	: 8

정수 데이터 처리 시 자료형 선택 기준

- ▶ 일반적인 선택은 **int**
 - CPU의 기본 연산이 int형임. 가장 빠르게 처리
 - 정수 연산 시 int로 형변환 후 연산 수행
 - int형 범위를 넘는 경우 → long형 사용
 - 연산을 수반하지 않으면서 많은 데이터(작은 수들로 이루어진)를 처리해야 한다면 → char, short 사용 가능

```
int main(void)
{
    char num1 = 1, num2 = 2, result1 = 0;
    short num3 = 300, num4 = 400, result2 = 0;
```

```
    printf("size of num1 & num2: %d, %d \n", sizeof(num1), sizeof(num2));
    printf("size of num3 & num4: %d, %d \n", sizeof(num3), sizeof(num4));
```

```
    printf("size of char add: %d \n", sizeof(num1 + num2));
    printf("size of short add: %d \n", sizeof(num3 + num4));
```

```
    result1 = num1 + num2;
    result2 = num3 + num4;
    printf("size of result1 & result2: %d, %d \n", sizeof(result1), sizeof(result2));
```

```
}
```

size of num1 & num2: 1, 1
 size of num3 & num4: 2, 2
 size of char add: 4
 size of short add: 4
 size of result1 & result2: 1, 21

int(4바이트)로 형
 변환 후 연산 수행

실수 데이터 처리 시 자료형 선택 기준

- ▶ 정밀도 고려 : 가수로 표현 가능한 소수점 이하 자리수

실수 자료형	소수점 이하 정밀도	바이트 수
float	6자리	4
double	15자리	8
long double	18자리	12

- 일반적으로는 **double** 사용

```
int main(void)
{
    double rad;
    double area;
    printf("원의 반지름 입력: ");
    scanf("%lf", &rad);

    area = rad * rad * 3.1415;
    printf("원의 넓이: %f \n", area);
}
```

원의 반지름 입력: 2.4
원의 넓이: 18.095040

unsigned 타입 : 0과 양의 정수만 표현

▶ unsigned 타입

- MSB(최상위 비트)까지도 부호가 아닌 데이터 표현에 사용
- 정수형 자료형에만 사용 가능

```
int main(void)
{
    unsigned char a = -128;
    printf("%d\n", a);    // ?
}
```

정수 자료형	크기	값의 표현범위
char	1바이트	-128이상 +127이하
unsigned char		0이상 (128 + 127)이하
short	2바이트	-32,768이상 +32,767이하
unsigned short		0이상 (32,768 + 32,767)이하
int	4바이트	-2,147,483,648이상 +2,147,483,647이하
unsigned int		0이상 (2,147,483,648 + 2,147,483,647)이하
long	4바이트	-2,147,483,648이상 +2,147,483,647이하
unsigned long		0이상 (2,147,483,648 + 2,147,483,647)이하
long long	8바이트	-9,223,372,036,854,775,808이상 +9,223,372,036,854,775,807이하
unsigned long long		0이상 (9,223,372,036,854,775,808 + 9,223,372,036,854,775,807)이하

문자 표현을 위한 약속! 아스키(ASCII) 코드

▶ ASCII 코드

- 미국 표준 협회(ANSI)에서 제정
- 내부적으로 정수로 표현
 - 65를 정수로 해석하면 65
 - 65를 문자로 해석하면 A

아스키 코드	아스키 코드 값
A	65
B	66
C	67
\	96
~	126

```
int main(void)
{
    char ch1 = 'A';
    char ch2 = 'C';
    . . . .
}
```



```
int main(void)
{
    char ch1 = 65;
    char ch2 = 67;
    . . . .
}
```

컴파일 시 각 문자는
해당 아스키 코드 값
으로 변환
- 실제 저장된 값은
정수값

C 프로그램에서 문자의 표현 :
작은 따옴표 내에 표기

문자 사용 예

```
int main(void)
{
    char ch1 = 'A', ch2 = 65;
    int  ch3 = 'Z', ch4 = 90;

    printf("%c %d \n", ch1, ch1);
    printf("%c %d \n", ch2, ch2);
    printf("%c %d \n", ch3, ch3);
    printf("%c %d \n", ch4, ch4);
}
```

A 65
A 65
Z 90
Z 90

%c : 해당 정수값을 아스키 코드 문자로 출력

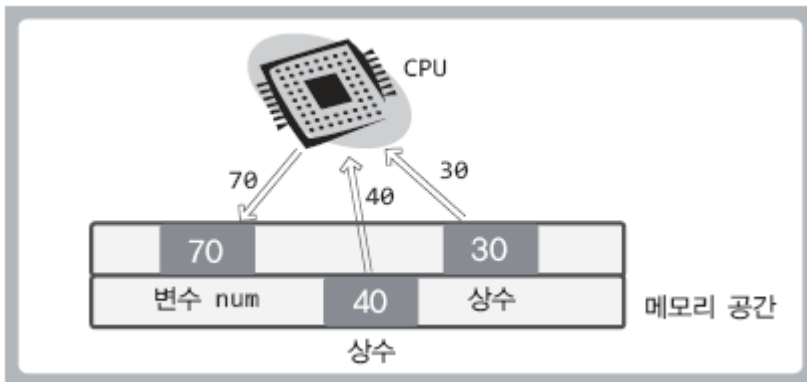
▶ ASCII 코드의 범위

- 0~127 : 총 128개
- 1바이트로 충분히 표현 가능 → char형 변수 사용
 - 그러나 int형 변수에도 저장 가능하며 주로 int를 많이 사용
 - 문자열(문자의 묶음)을 처리할 때는 char형 사용

이름을 지니지 않는 리터럴 상수!

- ▶ 상수 : 변경 불가능한 데이터
- ▶ 리터럴(literal) 상수 : 이름을 지니지 않는 상수
 - 30, 40, 3.4, 'A', ...
 - 상수도 메모리 공간을 차지함

```
int main(void)
{
    int num = 30 + 40;
    . . .
}
```



- 단계 1. 정수 30과 40이 메모리 공간에 상수의 형태로 저장된다.
- 단계 2. 두 상수를 기반으로 덧셈이 진행된다.
- 단계 3. 덧셈의 결과로 얻어진 정수 70이 변수 num에 저장된다.

메모리 공간에 저장되어야 CPU의 연산 대상이 됨

리터럴 상수의 자료형

- ▶ 상수도 자료형이 있음 → 해당 타입의 메모리 공간 차지
 - 5 : int
 - 3.15 : double
 - 'A' : int (문자는 int 상수 → 필요 시 char로 자동 형변환)

```
int main(void)
{
    printf("literal int size: %d \n", sizeof(7));
    printf("literal double size: %d \n", sizeof(7.14));
    printf("literal char size: %d \n", sizeof('A'));
}
```

```
literal int size: 4
literal double size: 8
literal char size: 4
```

접미사를 이용한 다양한 상수의 표현

```
int main(void)
{
    float num1 = 5.789;    // 경고 메시지 발생
    float num2 = 3.24 + 5.12; // 경고 메시지 발생
    return 0;
}
```

5.789는 double형 → float 변환 시
데이터 손실에 대한 경고 메시지 발생

```
float num1 = 5.789f;    // 경고 메시지 발생 안 함
float num2 = 3.24F + 5.12F; // 소문자 f 대신 대문자 F를 써도 된다!
```

접미사를 통해 상수의
자료형 변경 가능

접미사	자료형	사용의 예
U	unsigned int	unsigned int n = 1025U
L	long	long n = 2467L
UL	unsigned long	unsigned long n = 3456UL
LL	long long	long long n = 5768LL
ULL	unsigned long long	unsigned long long n = 8979ULL

[표 05-4: 정수형 상수의 표현을 위한 접미사]

접미사	자료형	사용의 예
F	float	float f = 3.15F
L	long double	long double f = 5.789L

[표 05-5: 실수형 상수의 표현을 위한 접미사]

이름을 지니는 심볼릭(Symbolic) 상수

▶ 심볼릭 상수 = const 상수

```
int main(void)
{
    const int MAX=100;    // MAX는 상수! 따라서 값의 변경 불가!
    const double PI=3.1415; // PI는 상수! 따라서 값의 변경 불가!
    . . . . .
}
```

```
int main(void)
{
    const int MAX;    // 쓰레기 값으로 초기화 되어버림
    MAX=100;    // 값의 변경 불가! 따라서 컴파일 에러 발생!
    . . . . .
}
```

- 선언 시 반드시 초기화
- 값 변경 불가능

- 상수의 이름 : 관례적으로 대문자 사용 (코딩 스타일)
- 둘 이상의 단어 연결 시 MY_AGE와 같이 언더바(_) 사용

(참고) 심볼릭 상수처럼 사용 가능한 #define

▶ #define 전처리문

- 전처리 과정에서 해당 문자열을 오른쪽 문자들로 대체하는 개념 // 문서 편집기에서 찾아바꾸기(replace)와 동일

```
#define MAX (1 + 2)
#define PI 3.1415

int main(void)
{
    double area = PI * MAX * MAX;

    printf("면적 : %f \n", area);
}
```

3.1415 * (1 + 2) * (1 + 2) 와 동일

면적 : 28.273500

C 언어에서는 #define 전처리문을 매우 자주 사용함
→ 보다 구체적인 사용 방법은 13주차에서 설명

대입 연산 시 발생하는 자동 형변환

▶ 정수 \leftrightarrow 실수 사이의 자동 형변환

대입 연산자의 왼쪽을
기준으로 형변환 발생

```
double num1=245;    // int형 정수 245를 double형으로 자동 형 변환
int num2=3.1415;    // double형 실수 3.1415를 int형으로 자동 형 변환
```


정수 245는 245.0의 비트 열로 재구성되어 변수 num1에 저장됨

실수 3.1415는 int형 데이터 3의 비트 열로 재구성되어 변수 num2에 저장됨

▶ 정수 사이의 자동 형변환

```
int num3=129;
char ch=num3;    // int형 변수 num3에 저장된 값이 char형으로 자동 형 변환
```

4바이트 변수 num3에 저장된 4바이트 데이터 중 상위 3바이트가 손실되어 변수 ch에 저장

00000000 00000000 00000000 10000001  10000001

자동 형변환의 방식 정리

- ▶ 대표적인 자동 형변환 사례
 - 정수와 실수 사이의 형변환
 - 정수를 실수로 형변환 : 3이 3.0으로 변경됨
 - 실수를 정수로 형변환 : 3.4 → 3, 소수점 이하값 소멸(그냥 버림)
 - 정수 사이의 형 변환, 실수 사이의 형 변환

```
int main(void)
{
    double num1 = 245;
    int num2 = 3.1415;
    int num3 = 129;
    char ch = num3;

    printf("정수 245를 실수로: %f \n", num1);
    printf("실수 3.1415를 정수로: %d \n", num2);
    printf("큰 정수 129를 작은 정수로: %d \n", ch);
}
```

정수 245를 실수로: 245.000000
실수 3.1415를 정수로: 3
큰 정수 129를 작은 정수로: -127

정수의 승격에 의한 자동 형변환

- ▶ 일반적으로 CPU가 처리하기에 가장 적합한 정수 타입
 - int
 - 따라서 정수 연산 시 가능하다면 int로 변환 후 처리
→ 정수의 승격(Integral Promotion)

```
int main(void)
{
    short num1 = 15, num2 = 25;
    short num3 = num1 + num2; // num1과 num2가 int형으로 변환됨

    printf("바이트 크기 : %d\n", sizeof(num1 + num2));
}
```

바이트 크기 : 4

피연산자의 자료형 불일치 → 자동 형변환

- ▶ 두 피연산자의 자료형은 일치해야 함
 - 일치하지 않으면 하나를 자동 형변환을 통해 일치시킴

```
double num1 = 5.15 + 19;
```

19(int)를 19.0(double)으로 변환

- ▶ 산술 연산에서의 자동 형변환 규칙
 - 데이터의 손실을 최소화하기 위해
 - 정수형보다 실수형 우선
 - 바이트 크기가 큰 자료형 우선

- 가능한 방향으로 적용
- 데이터의 손실을 최소화



명시적 형변환 : 강제로 일으키는 형변환

- ▶ 형변환의 종류 : 자동 형변환, 명시적 형변환
 - 명시적 형변환 : 프로그래머의 요청에 의한 형변환

```
int main(void)
{
    int num1 = 3, num2 = 4;
    double div_result;

    div_result = num1 / num2;
    printf("나눗셈 결과: %f \n", div_result);
}
```

나눗셈 결과: 0.000000

`div_result = (double) num1 / num2;`

```
int main(void)
{
    int num1 = 3;
    double num2 = 2.5 * num1;
    . . . .
}
```



```
int main(void)
{
    int num1 = 3;
    double num2 = 2.5 * (double)num1;
    . . . .
}
```

추천하는 코드 작성 스타일

자동 형변환이 발생하는 위치에 명시적 형변환 표시를 해서 형변환이 발생함을 알리는 것이 좋다!

이번 장에서 배운 것

- C 언어가 제공하는 기본 자료형에는 크게 정수형과 실수형이 있다.
- 정수형에는 char, short, int, long이 있고, unsigned를 붙여 0과 양의 정수만 표현할 수도 있다.
- 실수형에는 float, double, long double이 있다.
- 자료형 별로 사용하는 바이트 수가 다르며 표현 범위도 다르다.
- 문자 표현 방식으로는 아스키(ASCII) 코드를 사용한다.
- 상수는 변하지 않는 데이터이며, 상수도 타입이 있다.
- 리터럴 상수로는 'A', 3, 3.4 등이 있다.
- 심볼릭 상수는 const 키워드를 사용하여 만들 수 있으며 선언 시 반드시 초기값을 대입해야 한다.
- 형변환에는 자동 형변환과 명시적 형변환이 있다.
- 대입, 연산 시 자동 형변환 규칙에 따라 자동 형변환이 일어난다.