

3주차 2차시 상수와 자료형

【학습목표】

1. 문자 표현방식과 문자를 위한 자료형에 대해 설명할 수 있다.
2. 자료형의 변환에 대해 설명할 수 있다.

학습내용1 : 문자 표현방식과 문자를 위한 자료형

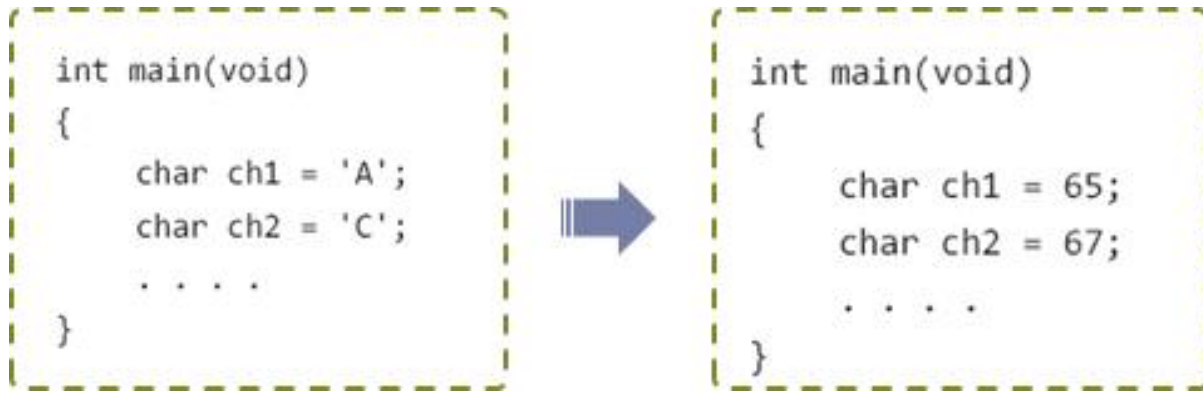
1. 문자의 표현을 위한 약속! 아스키(ASCII) 코드

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

√미국 표준 협회(ANSI: American National Standards Institute)에 의해서 제정된 '아스키(ASCII: American Standard Code for Information Interchange) 코드'

√컴퓨터는 문자를 표현 및 저장하지 못한다. 따라서 문자를 표현을 목적으로 각 문자에 고유한 숫자를 지정한다.

√인간이 입력하는 문자는 해당 문자의 숫자로 변환이 되어 컴퓨터에 저장 및 인식이 되고, 컴퓨터에 저장된 숫자는 문자로 변환이 되어 인간의 눈에 보여지게 된다



✓컴파일 시 각 문자는 해당 아스키 코드값으로 변환.

- 따라서 실제로 컴퓨터에게 전달되는 데이터는 문자가 아닌 숫자이다.

✓C 프로그램상에서 문자는 작은 따옴표로 묶어서 표현

2. 문자는 이렇게 표현되는 거구나!

```

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);
    return 0;
}

```

```

A 65
A 65
Z 90
Z 90

```

✓서식문자 %c는 해당 숫자의 아스키코드 문자를 출력하라는 의미

✓문자를 char형 변수에 저장하는 이유

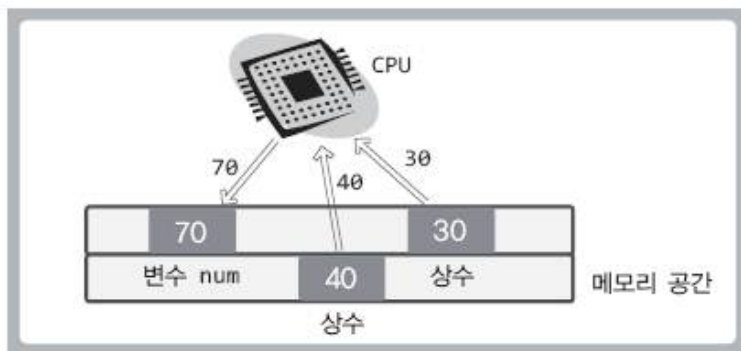
- 모든 아스키 코드 문자는 1바이트로도 충분히 표현가능
- 문자는 덧셈, 뺄셈과 같은 연산을 동반하지 않는다. 단지 표현에 사용될 뿐이다.
따라서 1바이트 크기인 char형 변수가 문자를 저장하기 최적의 장소이다.
문자는 int형 변수에도 저장이 가능하다.

학습내용2 : 상수에 대한 이해

1. 이름을 지니지 않은 리터럴 상수!

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

√연산을 위해서는 30, 40과 같이 프로그램상에 표현되는 숫자도 메모리 공간에 저장되어야 한다.
이렇게 저장되는 값은 이름이 존재하지 않으니 변경이 불가능한 상수이다.
따라서 리터럴 상수라 한다.



메모리 공간에 저장되어야 CPU의 연산대상이 된다.

단계1: 정수 30과 40이 메모리 공간에 상수의 형태로 저장된다.

단계2: 두 상수를 기반으로 덧셈이 진행된다.

단계3: 덧셈의 결과로 얻어진 정수 70이 변수 num에 저장된다.

2. 리터럴 상수의 자료형

```
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'));
    return 0;
}
```

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

✓ 리터럴 상수도 자료형이 결정되어야 메모리 공간에 저장이 될 수 있다.

위 예제의 실행결과는 다음 사실을 의미한다.

- 정수는 기본적으로 int형으로 표현된다.
- 실수는 기본적으로 double형으로 표현된다.
- 문자는 기본적으로 char형으로 표현된다.

3. 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;
}
```

실수는 double형 상수로 인식이 되어 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 통해서 상수의 자료형을 변경할 수 있다.

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]

4. 이름을 지니는 심볼릭(Symbolic) 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!
    . . . .
}
```

✓상수의 이름은 모두 대문자로 표시하고, 둘 이상의 단어를 연결할 때에는 MY_AGE와 같이 언더바를 이용해서 두 단어를 구분하는 것이 관례.

학습내용3 : 자료형의 변환

1. 대입 연산의 과정에서 발생하는 자동 형 변환

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

✓대입연산자의 왼쪽을 기준으로 형 변환이 발생한다.

정수 245는 245.0의 비트 열로 재구성이 되어 변수 num1에 저장된다.

실수 3.1415는 int형 데이터 3의 비트 열로 재구성이 되어 변수 num2에 저장된다.

```
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 저장된다.

00000000 00000000 00000000 10000001 ➡ 10000001

2. 자동 형 변환의 방식 정리

√형 변환의 방식에 대한 유형별 정리

- 정수를 실수로 형 변환 3은 3.0으로 5는 5.0으로 (오차가 발생하게 된다)
- 실수를 정수로 형 변환 소수점 이하의 값은 소멸된다.
- 큰 정수를 작은 정수로 형 변환 작은 정수의 크기에 맞추어 상위 바이트가 소멸한다.

```
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);
    return 0;
}
```

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

3. 정수의 승격에 의한 자동 형 변환

✓일반적으로 CPU가 처리하기에 가장 적합한 크기의 정수 자료형을 int로 정의한다.
따라서 int형 연산의 속도가 다른 자료형의 연산속도에 비해서 동일하거나 더 빠르다

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

이를 가리켜 '정수의 승격(Integral Promotion)'이라 한다.

4. 피연산자의 자료형 불일치로 발생하는 자동 형 변환

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

✓두 피연산자의 자료형은 일치해야 한다.
일치하지 않으면 일치하기 위해서 자동으로 형 변환이 발생한다.

✓아래의 자동 형 변환 규칙을 근거로 int형 데이터 19가 double형 데이터 19.0으로 형 변환이 되어 덧셈이 진행된다.



✓산술연산에서의 자동 형 변환 규칙

- 바이트 크기가 큰 자료형이 우선시 된다.
- 정수정보보다 실수형을 우선시 한다.
- 이는 데이터의 손실을 최소화 하기 위한 기준이다.

5. 명시적 형 변환: 강제로 일으키는 형 변환

```
int main(void)
{
    int num1=3, num2=4;
    double divResult;
    divResult = num1 / num2;
    printf("나눗셈 결과: %f \n", divResult);
    return 0;
}
```


나눗셈 결과: 0.000000

√ num1과 num2가 정수이기 때문에 몫만 반환이 되는 정수형 나눗셈이 진행

```
divResult = (double)num1 / num2;
```

(type)은 type형으로의 형 변환을 의미한다.

√ num1이 double형으로 명시적 형 변환 그리고 num1과 num2의 / 연산 과정에서의 산술적 자동 형 변환! 그 결과 실수형 나눗셈이 진행되어 divResult에는 0.75가 저장된다.

<pre>int main(void) { int num1 = 3; double num2 = 2.5 * num1; }</pre>		<pre>int main(void) { int num1 = 3; double num2 = 2.5 * (double)num1; }</pre>
---	---	---

√ 자동 형 변환이 발생하는 위치에 명시적 형 변환 표시를 해서 형 변환이 발생함을 알리는 것이 좋다!

【학습정리】

1. 문자를 표현하는 아스키코드 값은 0이상 127이하로 이루어져 있다.
2. char형은 문자의 표현을 목적으로 정의된 자료형이기 때문에 문자형으로 분류하기도 하지만 char형은 정수형이다.
3. 자료형의 변환이라는 것은 데이터의 표현방식을 바꾸는 것이다.
4. 피연산자의 자료형이 일치하지 않아서 발생하는 자동 형 변환은 데이터의 손실을 최소화하는 방향으로 진행된다.