기초프로그래밍

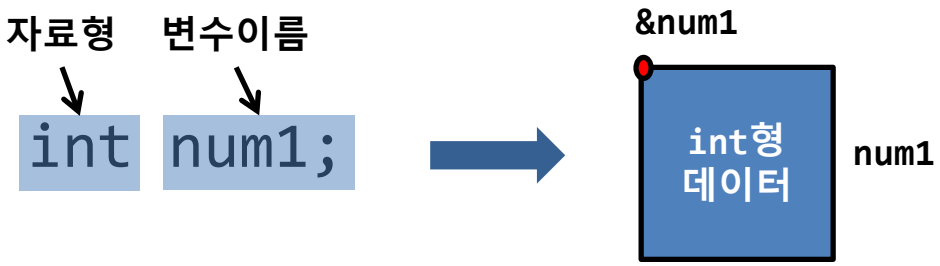
제6장 자료형

Sangsoo Lim
CSAI
Dongguk University

차례

자료형이란

- **자료형**
  - 변수가 저장하는 데이터 형식



- **자료형의 종류**
  - **정수형:** 정수를 표현하는 데이터 타입
  - **실수형:** 소수점이 포함된 값을 표현하는 데이터 타입

정수형				실수형		
char	short	int	long	float	double	long double
문자형						

자료형이란

- **자료형의 크기**
  - **sizeof 연산자: 자료형의 크기를 구하는 연산자**

사용법	예	설명
sizeof(자료형)	printf("%d", sizeof(int));	자료형의 메모리 크기를 출력
sizeof(변수)	int num1 = 3; printf("%d", sizeof(num1));	변수의 메모리 크기를 출력

  - **sizeof 연산자의 장점**
    - **자료형에 할당되는 메모리의 크기를 구할 수 있다.**

자료형이란

```c
/* 6-1.c */
char num1=10;
short num2=20;
int num3=30;
long num4=40;

printf("\n--------정수형 자료형과 변수의 메모리 크기--------\n");
printf("char형의 크기  %d바이트, %d바이트 \n", sizeof(char), sizeof(num1) );
printf("short형의 크기  %d바이트, %d바이트 \n", sizeof(short), sizeof(num2) );
printf("int형의 크기  %d바이트, %d바이트 \n", sizeof(int), sizeof(num3) );
printf("long형의 크기  %d바이트, %d바이트 \n", sizeof(long), sizeof(num4) );
```

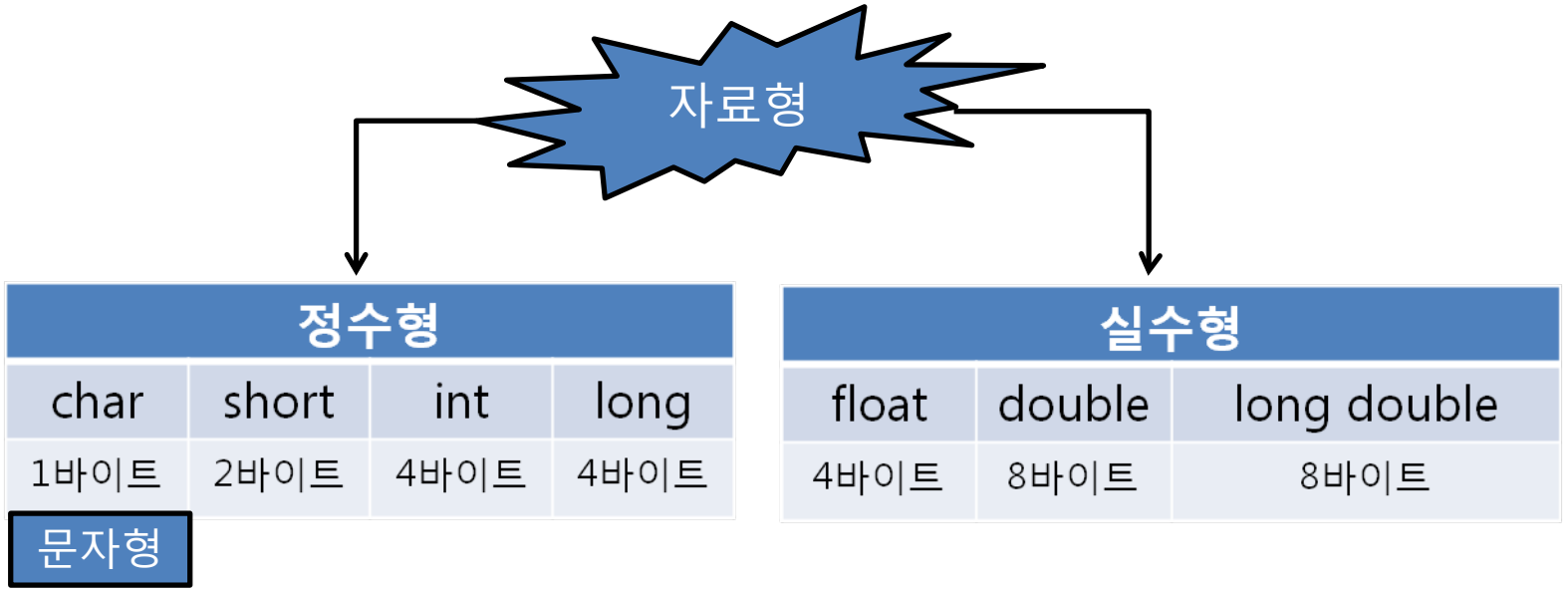
정수형			
char	short	int	long
1바이트	2바이트	4바이트	4바이트

```c
float num5=3.14;
double num6=3.15;
long double num7=3.17;

printf("\n--------실수형 자료형과 변수의 메모리 크기--------\n");
printf("float형의 크기 %d바이트, %d바이트 \n", sizeof(float), sizeof(num5) );
printf("double형의 크기 %d바이트, %d바이트 \n", sizeof(double), sizeof(num6) );
printf("long double형의 크기 %d바이트, %d바이트 \n", sizeof(long double), sizeof(num7) );
```

실수형		
float	double	long double
4바이트	8바이트	8바이트

자료형이란

- **기본 자료형의 메모리 크기**



정수형			
char	short	int	long
1바이트	2바이트	4바이트	4바이트

문자형

실수형		
float	double	long double
4바이트	8바이트	8바이트

정수형

- **정수형 종류**
  - char(1바이트), short(2바이트), int(4바이트), long(4바이트)

정수형	메모리 크기	데이터 표현 범위
char	1바이트(8비트)	-128 ~ +127
short	2바이트(16비트)	-32768 ~ +32767
int	4바이트(32비트)	-2147483648 ~ +2147483647
long	4바이트(32비트)	-2147483648 ~ +2147483647

  - **데이터의 표현 범위**

n은 비트 수(1바이트는 8비트)

-2^{n-1} ~ $+2^{n-1}-1$

최솟값(MIN) 최댓값(MAX)

정수형

- 정수형 데이터 표현 범위를 정의한 상수
  - limits.h : 정수형 데이터 표현 최솟값(MIN)과 최댓값(MAX) 상수 제공

정수형	메모리 크기	데이터 표현 범위
char	1바이트(8비트)	-128 ~ +127
short	2바이트(16비트)	-32768 ~ +32767
int	4바이트(32비트)	-2147483648 ~ +2147483647
long	4바이트(32비트)	-2147483648 ~ +2147483647

정수형	상수(최솟값)	상수(최댓값)
char	CHAR_MIX	CHAR_MAX
short	SHRT_MIX	SHRT_MAX
int	INT_MIX	INT_MAX
long	LONG_MIX	LONG_MAX

정수형

```c
/* 6-2.c */
#include <stdio.h>
#include <limits.h>  // 정수형의 최솟값(MIN), 최댓값(MAX) 상수 정의
int main(void)
{
   printf("char의 최솟값 %d, 최댓값 %d \n", CHAR_MIN, CHAR_MAX);
   printf("short의 최솟값 %d, 최댓값 %d \n", SHRT_MIN, SHRT_MAX);
   printf("int의 최솟값 %d, 최댓값 %d \n", INT_MIN, INT_MAX);
   printf("long의 최솟값 %ld, 최댓값 %ld \n", LONG_MIN, LONG_MAX);

   return 0;
}
```

정수형	메모리 크기	데이터 표현 범위
char	1바이트(8비트)	-128 ~ +127
short	2바이트(16비트)	-32768 ~ +32767
int	4바이트(32비트)	-2147483648 ~ +2147483647
long	4바이트(32비트)	-2147483648 ~ +2147483647

정수형

- 정수형의 양수 범위를 **두 배**로 늘리는 unsigned 자료형이 있다.
  - unsigned: 0과 양수만을 표현

정수형	메모리 크기	데이터 표현 범위
unsigned char	1바이트(8비트)	0 ~ (127 + 128)
unsigned short	2바이트(16비트)	0 ~ (32767 + 32768)
unsigned int	4바이트(32비트)	0 ~ (2147483647 + 2147483648)
unsigned long	4바이트(32비트)	0 ~ (2147483647 + 2147483648)

정수형

```c
/* 6-3.c */
#include <stdio.h>
int main(void)
{
    signed char num1=130;     // -128 ~ 127의 데이터 표현 범위
    unsigned char num2=130;   // 0 ~ 256의 데이터 표현 범위

    printf("%d \n", num1);    // -126 출력
    printf("%u \n", num2);    // 130 출력

    return 0;
}
```

```
-126
130
```

정수형

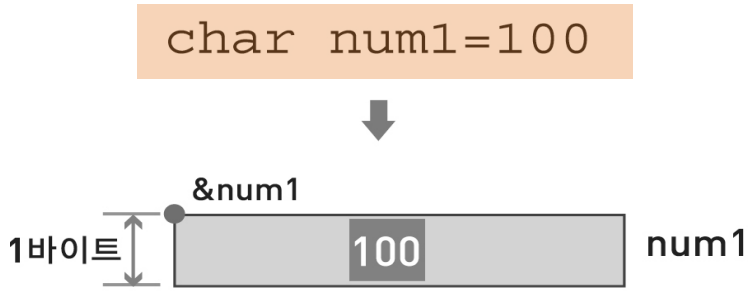
– limits.h 에서 제공하는 unsigned형 상수의 최댓값(MAX)

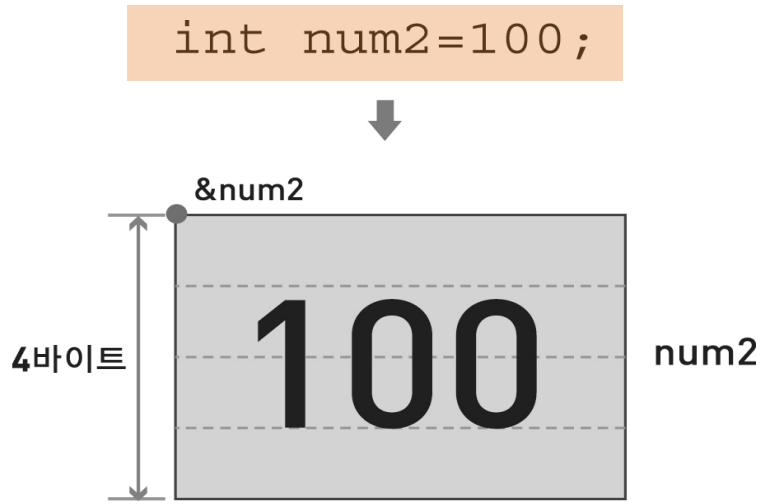
unsigned 정수형	상수(최댓값)
unsigned char	UCHAR_MAX
unsigned short	USHRT_MAX
unsigned int	UINT_MAX
unsigned long	ULONG_MAX

unsigned 정수형	데이터 표현 범위
unsigned char	0 ~ (127 + 128)
unsigned short	0 ~ (32767 + 32768)
unsigned int	0 ~ (2147483647 + 2147483648)
unsigned long	0 ~ (2147483647 + 2147483648)

정수형

- **정수형은 int형을 선호한다.**
  - **char형 변수와 int형 변수의 차이**

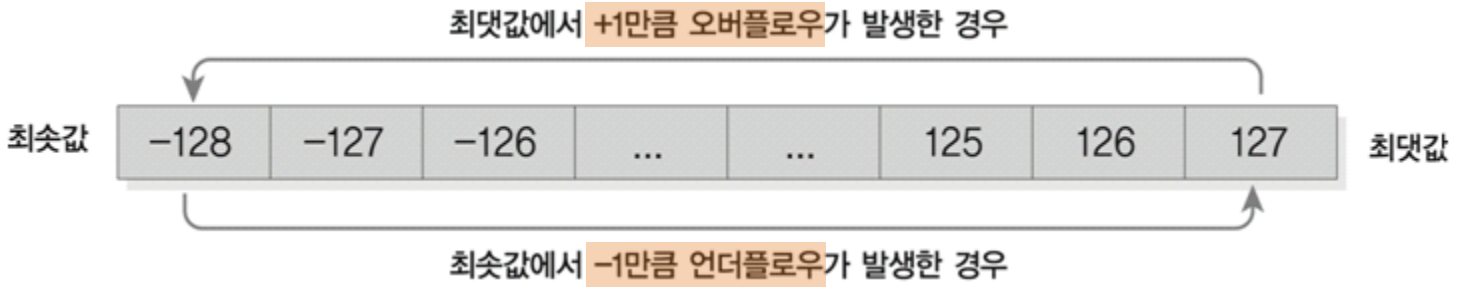




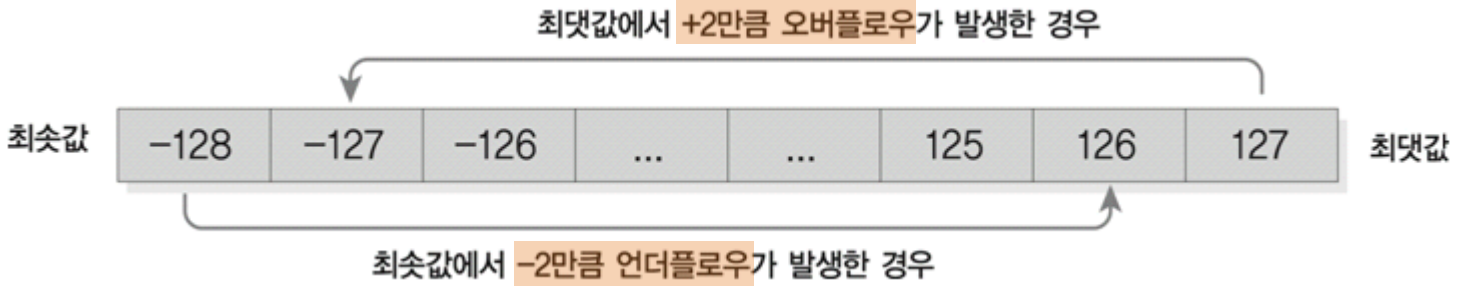
  - **CPU가 int형을 가장 빠르게 처리하는 이유**
    - 대부분의 컴퓨터들은 32비트 이상의 시스템
    - CPU가 연산하는 기본 단위가 최소 32비트

정수형

- **정수형의 오버플로우와 언더플로우는 순환된 값을 출력한다.**
  - **오버플로우** : 자료형에 저장할 수 있는 최대 범위보다 큰 수 저장



  - **언더플로우** : 자료형에 저장할 수 있는 최소 범위보다 작은 수 저장

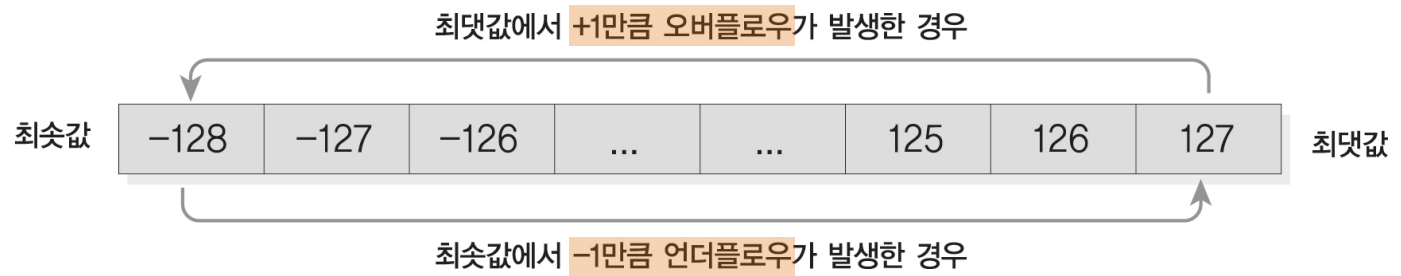


정수형

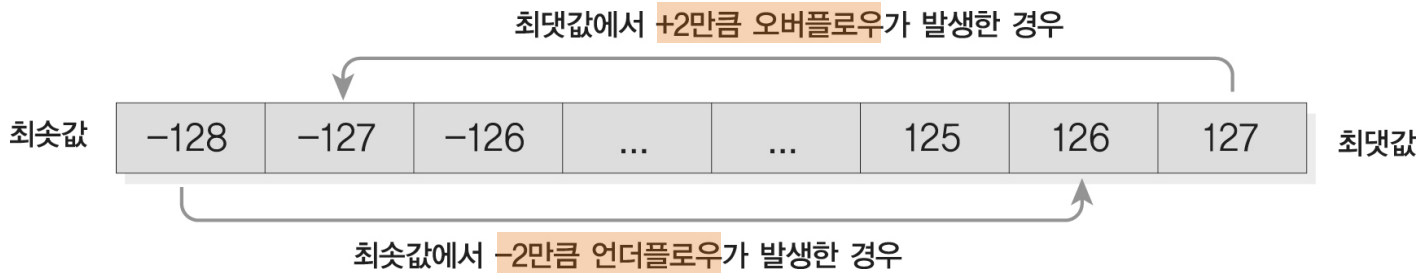
```c
/* 6-4.c */
#include <stdio.h>
int main(void)
{
    char num1=-129;   // 최솟값(-128)보다 -1만큼 작은 값 저장(언더플로우)
    char num2=128;    // 최댓값(127)보다 +1만큼 큰 값 저장(오버플로우)

    printf("%d \n", num1);   // 127 출력
    printf("%d \n", num2);   // -128출력

    num1=-130;        // 최솟값(-128)보다 -2만큼 작은 값 저장(언더플로우)
    num2=129;         // 최댓값(127)보다 +2만큼 큰 값 저장(오버플로우)

    printf("%d \n", num1);   // 126 출력
    printf("%d \n", num2);   // -127출력

    return 0;
}
```

정수형

```
char num1=-129;          // 최솟값보다 -1만큼 작은 값 저장(언더플로우)
char num2=128;           // 최댓값보다 +1만큼 큰 값 저장(오버플로우)
```




```
num1=-130;       // 최솟값(-128)보다 -2만큼 작은 값 저장(언더플로우)
num2=129;        // 최댓값(127)보다 +2만큼 큰 값 저장(오버플로우)
```



실수형

- **실수형이란**
  - 실수형 데이터를 저장하는 변수의 자료형
  - 소수점을 가진 실수의 값을 표현할 수 있는 자료형

실수형	메모리 크기	데이터 표현 범위
float	4바이트(32비트)	1.17×10^{-38} ~ 3.40×10^{38}
double	8바이트(64비트)	$2.22 * 10^{-308}$ ~ 1.79×10^{308}
long double	8바이트(64비트)	$2.22 * 10^{-308}$ ~ 1.79×10^{308}

실수형

- 실수형 데이터 표현 범위를 표현 범위를 정의한 상수
  - float.h : 실수형 데이터 표현 최솟값(MIN)과 최댓값(MAX) 상수 제공

```c
/* 6-5.c */
#include <stdio.h>
#include <float.h>    // 실수형의 데이터 표현 범위 상수 정의
int main(void)
{
   printf("float의 최솟값 %e, 최댓값 %e \n", FLT_MIN, FLT_MAX);
   printf("double의 최솟값 %e, 최댓값 %e \n", DBL_MIN, DBL_MAX);

   return 0;
}
```

```
float의 최솟값 1.175494e-38, 최댓값 3.402823e+38
double의 최솟값 2.225074e-308, 최댓값 1.797693e+308
```

실수형

```c
/* 6-6.c */
#include <stdio.h>
int main(void)
{
   float num1=3.4e+30;
   double num2=3.4e+30;

   printf("%f, %e \n", num1, num1);      // float형 오차발생
   printf("%lf, %le \n", num2, num2);    // double형은 정상

   return 0;
}
```

```
339999990004565769575209526886 4.000000, 3.400000e+30
340000000000000023649271047782 4.000000, 3.400000e+30
```

실수형

- 실수형은 데이터의 정밀도를 높이기 위해 사용
  - 99.9
  - 99.99
  - 99.999
  - 99.9999

실수형	표현 가능한 소수점 이하 자리 수
float	소수점 이하 6자리
double	소수점 이하 15자리
long double	소수점 이하 15자리 또는 그 이상

실수형

```c
/* 6-7.c */
#include<stdio.h>
int main(void)
{
   float num1=0.123456789012345;
   double num2=0.123456789012345;

   printf("float형 : %f \n", num1);      // 0.123457 출력
   printf("double형 : %lf \n", num2);    // 0.123457 출력

   printf("float형 : %.15f \n", num1);   // 0.123456791043282 출력
   printf("double형 : %.15lf \n", num2); // 0.123456789012345 출력

   return 0;
}
```

```
float형 : 0.123457
double형 : 0.123457
float형 : 0.123456791043282
double형 : 0.123456789012345
```

실수형

- **실수형은 double형을 선호한다.**
  - 오차를 줄이기 위해서 [6-6.c]
  - 정밀도를 높이기 위해서 [6-7.c]
  - 컴파일러는 기본적으로 실수형을 double로 인식 [6-8.c]

```c
/* 6-8.c */
#include<stdio.h>
int main(void)
{
   float num1=0.123456;                  // float num1=0.123456F;
   printf("float형 : %f \n", num1);    // 0.123456(소수점 6자리까지 출력)
   printf("float형 : %.2f \n", num1); // 0.12(소수점 2자리까지 출력)

   return 0;
}
```

문자형

- 컴퓨터(CPU)는 문자를 인식하지 못한다.
- 컴퓨터는 ASCII 코드를 참조해서 문자를 인식한다.
- 문자형은 char형을 선호한다.
  - 작은따옴표 안에 문자 하나를 입력 (ASCII에 지정된 숫자, 문자만 저장)
    - 사용 예) `char c = 'a';`
  - 잘못 사용한 문자형의 사례
    - 사용 예 1) `char c = '가';  // 한글은 2바이트`
    - 사용 예 2) `char c = a;  // 작은 따옴표가 없음`
    - 사용 예 3) `char c = "o";  // 큰 따옴표의 사용`

문자형

```c
/* 6-9.c */
#include <stdio.h>
int main(void)
{
   char val1;

   val1='A';
   printf("%d %c \n", val1, val1);   // 65  A 출력

   val1='B';
   printf("%d %c \n", val1, val1);   // 66  B 출력

   val1='C';
   printf("%d %c \n", val1, val1);   // 67  C 출력

   return 0;
}
```

문자형

```c
/* 6-10.c */
#include <stdio.h>
int main(void)
{
   char val1;
   int val2;

   printf("문자 입력 :");
   scanf("%c", &val1);
   printf("ASCII 코드 값 %d입니다. \n", val1);

   printf("ASCII 코드 값 입력 :”);
   scanf("%d", &val2);
   printf("문자로 %c입니다.\n", val2);

   return 0;
}
```

자료형 변환

- 자료형 변환의 종류
  - 자동 형변환
    - 컴파일러가 자동 형변환 시킨다.
  - 강제 형변환
    - 프로그래머 강제 형변환 시킨다.

자료형 변환

- 컴파일러가 자동으로 형변환을 해준다. – 자동 형변환
  - 다른 자료형 간 산술 연산의 경우에 작은형에서 큰형으로 자동 형변환
    - 정수 + 실수 또는 실수 + 정수의 산술 연산을 하는 경우

```
/* 6-11.c */
#include <stdio.h>
int main(void)
{
   int num1=100;                  // 정수
   double num2=3.14;              // 실수

   printf("%lf \n", num1+num2);      // 정수 + 실수
   return 0;
}
```

  - 자료형 변환 우선순위 (작은형에서 큰형으로…)
    - char < int < long < float < double < long double

자료형 변환

- 컴파일러가 자동으로 형변환을 해준다. – 자동 형변환
  - 대입 연산을 하는 경우
    - 대입 연산자를 기준으로 오른쪽에서 왼쪽으로 자동 형변환

```c
/* 6-12.c */
#include <stdio.h>
int main(void)
{
    char num1=130;
    int num2=3.14;
    double num3=3;

    printf("%d, %d, %lf \n", num1, num2, num3);
    return 0;
}
```

```
-126, 3, 3.000000
```

자료형 변환

- 컴파일러가 자동으로 형변환을 해준다. – 자동 형변환
  - 데이터 손실이 없는 경우
    - 예) int형 (작은 자료형) 에서 double형(큰 자료형)으로 변환되는 경우
  - 데이터 손실이 있는 경우
    - 예) double형 (큰 자료형) 에서 int형(작은 자료형)으로 변환되는 경우
- 데이터 손실이 있는 경우, 데이터 손실을 최소화 하는 방법은?
  - 강제 형변환
  - 자동 형변환의 문제점 보완

자료형 변환

- **프로그래머가 강제로 형변환을 해준다 – 강제 형변환**
  - **이미 정의된 자료형을 강제적으로 다른 자료형으로 변환하는 것**
  - **괄호 연산자 ()를 이용**

```
int num1=2;
(double)num1;
```

자료형 변수

자료형 변환

```c
/* 6-13.c */
#include <stdio.h>
int main(void)
{
   int num1=10, num2=3;
   double result;

   result=num1/num2;
   printf("결과 : %lf \n", result);

   result=(double)num1/num2;
   printf("결과 : %lf \n", result);

   result=num1/(double)num2;
   printf("결과 : %lf \n", result);

   result=(double)num1/(double)num2;
   printf("결과 : %lf \n", result);
   return 0;
}
```

```
결과 : 3.000000
결과 : 3.333333
결과 : 3.333333
결과 : 3.333333
```

typedef를 이용한 자료형의 재정의

- 기본 자료형들에 새로운 이름을 붙이는 용도
  - 자료형을 간결하게 표현 가능
  - 프로그램의 가독성을 높일 수 있음
  - 너무 남용하면 자료형 분석 시 혼란 초래

```
typedef int mytype;
```

기본 자료형 사용자 정의 자료형

typedef를 이용한 자료형의 재정의

```c
/* 6-14.c */
#include <stdio.h>

typedef int money;

int main(void)
{
   money num1=3000;
   money num2=10000;
   money num3=2000;
   money num4=0;

   num4=num1+num2+num3+num4;
   printf("total money : %d won \n", num4);

   return 0;
}
```

Summary

- 자료형이란 무엇인가
- 정수 자료형
  - char형, short형, int형, long형
- 실수 자료형
  - float형, double형, long double형
- 자료형 변환
  - 자동 형변환
  - 강제 형변환
- typedef를 이용한 자료형 재정의

1. 자료형이란

- **예제 1-1: sizeof 활용하기**

```c
#include <stdio.h>

int main(void)
{
    printf("char     : %zu byte\n", sizeof(char));
    printf("short    : %zu bytes\n", sizeof(short));
    printf("int      : %zu bytes\n", sizeof(int));
    printf("long     : %zu bytes\n", sizeof(long));
    printf("float    : %zu bytes\n", sizeof(float));
    printf("double   : %zu bytes\n", sizeof(double));
    printf("long double : %zu bytes\n", sizeof(long double));

    return 0;
}
```

2. 정수형

- **예제 2-1: 오버플로우와 언더플로우 실습**

```c
#include <stdio.h>
#include <limits.h>

int main(void)
{
    char c = CHAR_MAX;  // char 최댓값 (127)
    printf("char 최댓값 : %d\n", c);

    c = c + 1;  // 오버플로우
    printf("오버플로우 후 : %d\n", c); // -128(시스템에 따라 다를 수 있음)

    unsigned char uc = 0;
    printf("unsigned char 초기값: %u\n", uc);

    uc = uc - 1;  // 언더플로우
    printf("언더플로우 후 : %u\n", uc);

    return 0;
}
```

3. 실수형

- **예제 3-1: 부동소수점 오차**

```c
#include <stdio.h>

int main(void)
{
    float f = 0.1f;
    double d = 0.1;

    // 같은 0.1이지만 내부 표현은 다를 수 있음
    if (f == d)
        printf("f와 d는 같습니다.\n");
    else
        printf("f와 d는 다릅니다.\n");

    printf("float f = %.20f\n", f);
    printf("double d = %.20f\n", d);

    return 0;
}
```

5. 자료형 변환(형변환)

- **예제 5-1: 분명한 형변환으로 정밀한 나눗셈**

```c
#include <stdio.h>

int main(void)
{
    int a = 3, b = 2;
    double result;

    // 자동 형변환 없이 정수 / 정수로 계산
    result = a / b;
    printf("정수 나눗셈 결과: %lf\n", result); // 결과는 1.000000

    // 강제 형변환 적용
    result = (double)a / b;
    printf("실수 나눗셈 결과: %lf\n", result); // 결과는 1.500000

    return 0;
}
```

6. typedef를 이용한 자료형 재정의

- **예제 6-1: 금액 자료형 정의**

```c
#include <stdio.h>

typedef unsigned long MONEY;

int main(void)
{
    MONEY price = 50000;
    MONEY discount = 10000;

    MONEY finalPrice = price - discount;
    printf("최종 금액: %lu원\n", finalPrice);

    return 0;
}
```

1. 자료형 범위 확인 및 오버플로우 테스터

1. 문제 설명
   - char, int, float, double 등의 최대·최소 범위를 출력하고, 특정 값 이상으로 입력했을 때 오버플로우/언더플로우가 일어나는지 확인하는 프로그램을 작성하라.

2. 요구사항
   - limits.h, float.h를 적절히 포함해 최대·최소값을 출력.
   - 사용자에게 정수/실수를 각각 입력받고, 변수에 저장했을 때 어떻게 변하는지 출력.

```
[ 정수형 범위 ]
char  : -128 ~ 127
int   : -2147483648 ~ 2147483647

[ 실수형 범위 ]
float : 1.175494e-38 ~ 3.402823e+38
double: 2.225074e-308 ~ 1.797693e+308

정수를 입력하세요: 2147483647
입력한 값: 2147483647, 저장된 값: 2147483647

실수를 입력하세요: 9999999999
입력한 값: 9999999872.000000, 저장된 값:
9999999872.000000
```

2. 형변환 실습 (정수/실수 계산기)

1. 문제 설명
   - 정수 두 개와 실수 한 개를 입력받아, 여러 가지 연산을 수행한 뒤 출력한다.
   - 연산 과정에서 정수형 계산, 실수형 계산을 혼합하여 자동 형변환, 강제 형변환을 각각 시도해본다.
2. 요구사항
   - 예: (double)(a + b) vs. (double)a + (double)b 결과 비교.
   - 정수/실수로 나누기((double) a / b) 등 여러 케이스.

```
정수 2개와 실수 1개를 차례로 입력하세요: 10 3 2.5

1) 정수 + 정수 = 13
2) 정수 + 실수(자동 형변환) = 12.500000
3) (double)(a+b) + c = 13.500000
4) a/b (자동)  = 3.000000
5) a/b (강제)  = 3.333333
```

```
정수 2개와 실수 1개를 차례로 입력하세요: 4 2 7.7

1) 정수 + 정수 = 6
2) 정수 + 실수(자동 형변환) = 11.700000
3) (double)(a+b) + c = 11.700000
4) a/b (자동)  = 2.000000
5) a/b (강제)  = 2.000000
```

3. typedef 활용 간단 프로젝트

1. 문제 설명
   - typedef를 사용하여 새로운 자료형 이름을 정의한다(예: typedef float SCORE;).
   - 학생의 성적을 입력받아 평균을 구하는 프로그램 작성
   - .사용자가 입력하는 점수는 실수로 가능(예: 85.5, 90.0 등).
2. 요구사항
   - SCORE라는 이름으로 float(또는 double)를 대체.
   - 여러 과목 성적을 더하고, 평균을 계산.
   - 평균을 적절히 형식 지정(소수점 이하 자리수)하여 출력.

```
국어, 영어, 수학 점수를 입력하세요 (예: 85.5 90.0 78.3):
85.5 90 78.3
총합: 253.80
평균: 84.60
```

```
국어, 영어, 수학 점수를 입력하세요 (예: 85.5 90.0 78.3):
95 100.0 88.7
총합: 283.70
평균: 94.57
```