3장 기본 자료형

선언, 수식, 배정

- 모든 변수는 사용되기 전에 반드시 선언되어야 함
- 일반적인 프로그램의 시작 형태

```
#include <stdio.h>
int main(void){
   int     a, b, c;                    /* declaration */
   float  x, y = 3.3, z = -7.7;   /* declaration with
                                      initialization */
   printf("Input two integers: "); /* function call */
   scanf("%d%d", &b, &c);           /* function call */
   a = b + c;                  /* assignment statement */
   x = y + z;                  /* assignment statement */
   ......
```

변수 선언의 목적

- 변수의 메모리 공간 확보
- 올바른 연산자 선택
  - int a, b, c;

    a = b + c;
  - float d, e, f;

    d = e + f;
  - 위의 두 +는 기호는 같지만,

    위의 +는 정수 +이고,

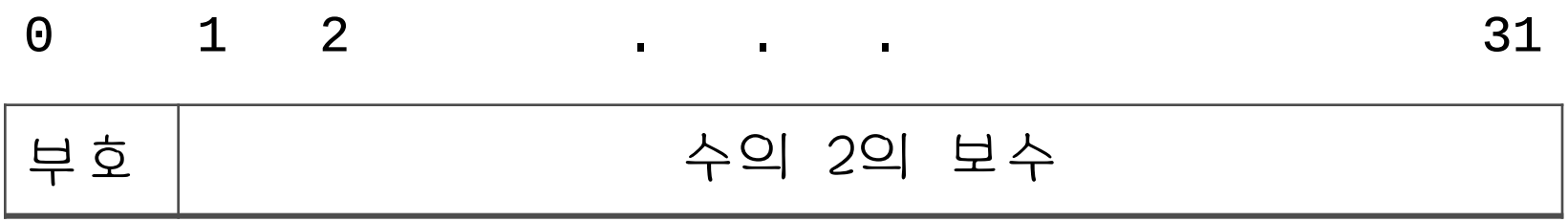
    밑의 +는 실수 +임

참조

- 정수와 실수 연산이 다른 이유
  - 컴퓨터에서 정수와 실수의 표현방법이 다르기 때문
- 정수
  - 부호가 있는 2의 보수로 표현
  - 1 비트 : 부호 비트 (1 : 음수, 0 : 양수)
  - 나머지 비트 : 수의 2의 보수

참조

- 정수의 표현



- 예

```
 2 : 00000000 00000000 00000000 00000010
-2 : 11111111 11111111 11111111 11111110
```

참조

- 실수
  - 부동소수로 표현
  - 모든 수를 지수 형태로 변환하여 부호, 지수, 가수부로 나누어 저장
  - IEEE 부동소수

참조

- 실수의 표현(IEEE 부동소수)

```
0  1 2 . . . 8 9    .    .    .          31
```

부호	지수부	가수부(1 hidden bit)

- 예

```
2.0  = 1.0 x 2^1 : 01000000 00000000 00000000 00000010
4.0  = 1.0 x 2^2 : 01000000 10000000 00000000 00000010
-2.0  = -1.0 x 2^1 : 11000000 00000000 00000000 00000010
```

블록, 수식, 문장

- 블록
  - 선언과 문장이 중괄호로 둘러싸져 있는 것
  - 선언은 반드시 문장 앞에 와야 함
- 수식
  - 상수, 변수, 연산자, 함수 호출 등의 의미 있는 결합
  - 상수, 변수, 함수 호출은 그 자체가 수식 임
  - 대부분의 수식은 값을 가짐

블록, 수식, 문장

- 문장
  - 수식 뒤에 세미콜론이 오면, 수식은 문장이 됨
- i = 7;
  - "i = 7"은 수식이고 그 값은 7임
  - "i = 7;"은 문장임

기본 자료형

- 완전한 형태

```
char                 signed char     unsigned char
signed short int     signed int      signed long int
unsigned short int   unsigned int    unsigned long int
float                double          long double
```

- 축약된 형태

```
char                 signed char     unsigned char
short                int             long
unsigned  short      unsigned        unsigned long
float                double          long double
```

기본 자료형

- 기본형의 기능별 분류
  - 정수적형

    ```
    char            signed char   unsigned char
    short           int           long
    unsigned short  unsigned      unsigned long
    ```

  - 부동형

    ```
    float           double        long double
    ```

  - 산술형

    정수적형 + 부동형

문자

- 모든 정수적형의 변수는 문자를 표현하는 데 사용될 수 있음
- 'a', '+'와 같은 상수는 char 형이 아니라 int 형임
  - C에는 char 형 상수가 없음
  - C++에서 문자 상수는 char 형임

char 자료형

- char 형 변수는 문자와 정수 값을 저장하는 데 사용됨
- 보통 각 char는 메모리의 1 바이트에 저장됨
  - 256개의 값을 저장할 수 있음
  - 이 값들 중 일부분만이 실제 인쇄 문자임
    (소문자, 대문자, 숫자, 마침표, 특수문자 등)
  - 공백, 탭, 개행 문자 같은 여백도 있음
- 대부분의 컴퓨터는 ASCII나 EBCDIC 문자 코드 사용

ASCII 코드

- ASCII 코드에서 문자 상수와 대응되는 정수 값

문자 상수	'a'	'b'	'c'	...	'z'
대응하는 값	97	98	99	...	122
문자 상수	'A'	'B'	'C'	...	'Z'
대응하는 값	65	66	67	...	90
문자 상수	'0'	'1'	'2'	...	'9'
대응하는 값	48	49	50	...	57
문자 상수	'&'	'*'	'+'		
대응하는 값	38	42	43		

ASCII 코드

- 주의
  - 숫자를 표현하는 문자 상수의 코드 값과 실제 그 숫자 값은 특별한 관계가 없음
    - 즉, '2'의 값은 2가 아님
  - 'a', 'b', 'c' 등의 코드 값은 순서적임
    - 이것은 문자와 단어를 사전적 순서로 정렬할 때 편리함

탈출 기법

- 인쇄할 수 없는 문자는 탈출 기법을 사용하여 표현함
  - 예를 들어, 수평 탭 문자는 문자 상수와 문자열에서 ₩t로 표현됨
  - ₩t가 ₩와 t 두 문자로 기술되지만, 이것은 한 문자임
- 프로그램 내에서 특별한 의미를 갖는 문자들이 본래의 의미를 갖기 위해서도 탈출 기법을 사용해야 함
  - 큰따옴표를 포함하는 "abc"라는 문자열은 "₩"abc₩""로 표기함
  - 작은따옴표 문자 상수 '는 '₩''로 표기함

특수 문자

문자 이름	기술 형태	정수 값
경고	\a	7
역슬래시	\\	92
백스페이스	\b	8
캐리지 리턴	\r	13
큰따옴표	\"	34
폼피드	\f	12
수평 탭	\t	9
개행	\n	10
널 문자	\0	0
작은따옴표	\'	39
수직 탭	\v	11

char 형

- 각 값은 메모리에 이진수로 저장됨

  예) char c = 'a';

  - 변수 c는 메모리 1 바이트에 01100001로 저장됨 (ASCII로 저장)
  - 이 값을 계산하면 97이 됨 (ASCII 값)

- ANSI C는 char, signed char, unsigned char를 제공

char 형

- char는 시스템에 따라 signed char 또는 unsigned char와 같음
- 세 가지의 char 형은 각각 1 바이트에 저장됨 (256개의 값 표현)
- signed char 형의 값의 범위 : -128 ~ 127
- unsigned char 형의 범위 : 0 ~ 255

예제 코드 I

```
char c = 'a';
printf("%c", c);            /* a is printed */
printf("%d", c);            /* 97 is printed */
printf("%c%c%c", c, c + 1, c + 2);
                            /* abc is printed */
```

예제 코드(2)

```
char  c;
int   i;
for (i = 'a'; i <= 'z'; ++i)
   printf("%c", i);
               /* abc . . . z is printed */
for (c = 65; c <= 90; ++c)
   printf("%c", c);
               /* ABC . . . Z is printed */
for (c = '0'; c <= '9'; ++c)
   printf("%d ", c);
               /* 48 49 . . . 57 is printed */
```

int 자료형

- C 언어의 기본적인 자료형
- int 형은 2 바이트(= 16 비트)나 4 바이트(= 32 비트)에 저장됨
  - 즉, 정수의 일부분만 저장할 수 있음

int 자료형

- 값의 범위
  - 4 바이트 워드 컴퓨터
    - 최소 = -2^{31} = -2147483648
    - 최대 = $+2^{31} - 1$ = +2147483647
  - 2 바이트 워드 컴퓨터
    - 최소 = -2^{15} = -32768
    - 최대 = $+2^{15} - 1$ = +32767

int 자료형

- 정수 오버플로
  - 값의 범위를 초과할 때 발생 (주의 필요)

```
#define BIG 2000000000      /* 2 billion */

int main(void){
    int a, b = BIG, c = BIG;
    a = b + c;              /* out of range? */
    .....
```

  - 정수 오버플로가 발생해도 프로그램은 계속 수행되지만, 논리적으로 부정확한 값이 계산됨

정수적형 short, long, unsigned

- short 자료형은 기억장소를 절약하고자 하는 경우에 사용
- long 형은 큰 정수 값을 다룰 때 사용
- short 형은 2 바이트에 저장되고, long 형은 4 바이트에 저장됨
  - 4 바이트 워드 컴퓨터: int 형과 long 형 크기 같음
  - 2 바이트 워드 컴퓨터: int 형과 short 형 크기 같음
- unsigned 형의 변수는 음수가 아닌 정수를 표현할 때 사용함

정수적형 short, long, unsigned

- 정수 상수에 그 형을 명시하기 위해서는 접미사를 붙일 수 있음
- 접미사가 붙지 않은 정수 상수의 형은 int, long, unsigned long 중 하나 임
  - 시스템은 세 가지 형 중 그 정수 상수를 표현할 수 있는 첫 번째 것을 선택하여 그 정수 상수의 형으로 함
  - 예를 들어, 2 바이트 워드 컴퓨터에서 상수 32000은 int 형이고, 33000은 long 형임

정수적형 short, long, unsigned

- 접미사

u 또는 U	unsigned	37U
l 또는 L	long	37L
ul 또는 UL	unsigned long	37UL

부동형

- 실수 값을 다루기 위해 사용함
- 정수를 부동형 상수로 표현할 때에는 소수점을 사용해야 함
  - 상수 1.0과 2.0은 double형이지만,
  - 상수 3은 int 형임
- 접미사

f 또는 F	float	3.7F
l 또는 L	long double	3.7L

부동형 상수 표기법

- 십진 표기법
  - 1.0 또는 1. 또는 .0001
- 지수 표기법
  - 1.234567e5 (= 1.234567×10^5)
- 올바른 부동형 상수
  - 3.14159, 314.159e-2F, 0e0, 1.

부동형 상수 표기법

- 잘못된 부동형 상수
  - 3.14,159 /* comma not allowed */
  - 314159 /* decimal point or exponential part needed */
  - .e0 /* integer part or fractional part needed */
  - -3.14159 /* this is a floating constant expression */

부동형의 값의 범위

- 부동형에 배정될 수 있는 값은 *정밀도*와 *범위*라는 속성으로 기술됨
  - 정밀도 : 부동형 값이 가질 수 있는 유효숫자 수
  - 범위 : 부동형의 변수에 저장될 수 있는 가장 큰 양수와 가장 작은 양수의 한계

부동형의 값의 범위

- float 형
  - 정밀도 : 대략 유효숫자 6자리
  - 범위 : 대략 10^{-38}에서 10^{+38}
- double 형
  - 정밀도 : 대략 유효숫자 15자리
  - 범위 : 대략 10^{-308}에서 10^{+308}

예제

- x = 123.45123451234512345;
  - x의 실제 값
    - double : $0.12345123451234512345 \times 10^3$
    - float : 0.123451×10^3
- 주의
  - 모든 실수를 다 표현할 수는 없다.
  - 정수 산술 연산과 달리 부동형 산술 연산은 정확하지 않다.

typedef

- typedef의 목적
  - 긴 선언문을 축약해 쓸 수 있음
  - 사용 목적에 맞게 형 이름을 결정할 수 있음
  - 프로그램 이식을 쉽게 할 수 있게 함
- 예제

```
typedef     char     uppercase;
typedef     int  INCHES, FEET;
uppercase   u;
INCHES       length, width;
```

sizeof

- 객체를 저장하는 데 필요한 바이트 수를 알아내기 위해 사용
- 다른 단항 연산자와 동일한 우선 순위와 결합 법칙을 가짐
- 사용법
  - sizeof(object)

sizeof

- 기본형들의 크기 비교

  sizeof(char) = 1

  sizeof(short) <= sizeof(int) <= sizeof(long)

  sizeof(signed) = sizeof(unsigned) = sizeof(int)

  sizeof(float) <= sizeof(double) <= sizeof(long double)

getchar()와 putchar()

- stdio.h에 정의된 매크로
  - getchar() : 키보드에서 문자를 읽음
  - putchar() : 화면에 문자를 출력함
- char 형은 1 바이트, int 형은 2 바이트나 4 바이트에 저장됨
  - int 형은 char 형이 저장할 수 있는 모든 값과 그 이상을 저장할 수 있음
  - char 형을 작은 정수형으로 생각할 수 있음
  - 반대로 int 형을 큰 문자형으로 볼 수 있음

getchar()와 putchar()

- 예제 프로그램

```
#include <stdio.h>
int main(void){
    int   c;
    while ((c = getchar()) != EOF) {
                                /* #define EOF (-1) */
        putchar(c);
        putchar(c);
    }
    return 0;
}
```

(주의) c가 char가 아니라 int로 선언되어 있음

예제

```
/* In file capitalize.c */
#include <stdio.h>
int main(void){
   int c;
   while ((c = getchar()) != EOF)
      if (c >= 'a' && c <= 'z')
         putchar(c + 'A' - 'a');
      else
         putchar(c);
   return 0;
}
```

수학 함수

- C에는 수학 함수가 내장되어 있지 않음
- 다음과 같은 수학 함수는 표준 라이브러리의 일부분인 수학 라이브러리로 제공됨

  sqrt() pow() exp() log()

  sin() cos() tan()
- 이러한 수학 함수를 사용하기 위해서는 <math.h>를 포함시켜야 함

수학 함수

- 정수용 abs()와 실수용 fabs()를 구별해서 사용해야 함
- UNIX에서는 수학 라이브러리가 표준 라이브러리가 아니기 때문에 수학 라이브러리를 사용하는 경우 컴파일할 때 명시해야 함

  cc *pgm.c* **-lm**

수학함수 예제 프로그램

```
#include <math.h>
#include <stdio.h>
int main(void){
   double   x;
   . . . .

  printf("\n%15s%22.15e\n%15s%22.15e\n%15s%22.
  15e\n\n",
          "x = ", x,
          "sqrt(x) = ", sqrt(x),
          "pow(x, x) = ", pow(x, x));
   . . . .
}
```

변환

- 산술 수식은 값과 형을 가짐
- 필요에 따라 수식을 구성하는 형들은 변환이 일어남
- 정수적 승격
  - signed/unsigned char, signed/unsigned short, 열거형, signed/unsigned int는 정수적 수식에 사용될 수 있음
  - 이 수식에서 모든 형들은 int나 unsigned int로 변환됨

변환

- 예

```
char c = 'A';
printf("%c\n", c);
```

  - printf에서 c는 정수적 승격이 일어나 int 형이 됨

일반적인 산술 변환

- 산술 변환은 이항 연산자의 피연산자를 계산할 때 일어남
- 규칙

  둘 중 하나가 long double 형이면, 다른 하나는 long double 형으로 변환
  아니면, 둘 중 하나가 double 형이면, 다른 하나는 double 형으로 변환
  아니면, 둘 중 하나가 float 형이면, 다른 하나는 float 형으로 변환
  아니면, 정수적 승격이 일어나고, 다음 규칙 사용:
    둘 중 하나가 unsigned long 형이면, 다른 하나는 unsigned long 형으로 변환
    아니면, 둘 중 하나가 long 형이고 다른 하나가 unsigned 형이면, 다음 수행:
      만일 long 형이 unsigned 형의 모든 값을 표현할 수 있다면, unsigned 형은 long 형으로 변환
      아니면, 두 피연산자는 unsigned long 형으로 변환
    아니면, 둘 중 하나가 long 형이면, 다른 하나는 long 형으로 변환
    아니면, 둘 중 하나가 unsigned 형이면, 다른 하나는 unsigned 형으로 변환
    아니면, 두 피연산자는 int 형을 가짐

일반적인 산술 변환

- 자동 변환
- 묵시적 변환
- 강압
- 승격
- 확대

자동 변환 예제

선언

```
char c;     short s;      int i;
long l;     unsigned u;   unsigned long ul;
float f;    double d;     long double ld;
```

수식	형	수식	형
c - s / i	int	u * 7 - i	unsigned
u * 2.0 - i	double	f * 7 - i	float
c + 3	int	7 * s * ul	unsigned long
c + 5.0	double	ld + c	long double
d + s	double	u - ul	unsigned long
2 * i / l	long	u – l	*system-dependent*

캐스트

- 명시적인 변환
- 캐스트 예제

```
(double) i
(long) ('A' + 1.0)
x = (float) ((int) y + 1)
(double) (x = 77)
```

- 잘못된 캐스트

```
(double) x = 77
/* equivalent to ((double) x) = 77, error */
```

16진 상수와 8진 상수

- C 언시 코드에서 0으로 시작하는 양의 정수 상수는 8진 정수이고
- 0x 또는 0X로 시작하는 양의 정수 상수는 16진 정수임

16진 상수와 8진 상수

```
#include <stdio.h>
int main(void) {
   printf("%d  %x  %o\n", 19, 19, 19);
                          /* 19  13  23  */
   printf("%d  %x  %o\n", 0x1c, 0x1c, 0x1c);
                          /* 28  1c  34  */
   printf("%d  %x  %o\n", 017, 017, 017);
                          /* 15    f  17 */
   printf("%d\n", 11 + 0x11 + 011);
                          /* 37          */
   printf("%x\n", 2097151);
                          /* 1fffff      */
   printf("%d\n", 0x1FfFFf);
                          /* 2097151     */
   return 0;
}
```