

C언어 강의자료

문정욱

C언어 더 알아보기 4

Overflow and Underflow

■ Signed type(2의 보수 표현)

- n bit 타입일 경우
 - 최대값: $2^{n-1}-1$
 - 최소값: -2^{n-1}
- Overflow
 - 최대값+1 → 최소값
- Underflow
 - 최소값-1 → 최대값

논리 오류의 일종

Underflow

Overflow

이진수	2의 보수
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

Overflow and Underflow

■ Overflow

0111 1111 1111 1111 +32767



1000 0000 0000 0000 -32768

■ Underflow

1000 0000 0000 0000 -32768



0111 1111 1111 1111 +32767

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

```
int main(void)
{
```

```
    short a;
```

```
    a=0x7FFF;
```

```
    printf("%d\n", a);
```

```
    printf("%d\n", ++a); // overflow
```

```
    a=0x8000;
```

```
    printf("%d\n", a);
```

```
    printf("%d\n", --a); // underflow
```

```
    return 0;
```

```
}
```

입출력 결과

32767

-32768

-32768

32767

계속하려면 아무 키나 누르십시오 . . .

Overflow and Underflow

■ Unsigned type

- n bit 타입일 경우
 - 최대값: $2^n - 1$
 - 최소값: 0
- Overflow
 - 최대값 + 1 → 최소값
- Underflow
 - 최소값 - 1 → 최대값

논리 오류의 일종

Underflow

Overflow

이진수	2의 보수
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Overflow and Underflow

■ Overflow



■ Underflow



```
#include <stdio.h>

int main(void)
{
    unsigned short a;

    a=0xFFFF;
    printf("%d\n", a);
    printf("%d\n", ++a); // overflow

    a=0x0000;
    printf("%d\n", a);
    printf("%d\n", --a); // underflow
    return 0;
}
```

입출력 결과

```
65535
0
0
65535
계속하려면 아무 키나 누르십시오 . . .
```

배열

■ 배열의 타입

`X a[N]; // T(a) == X[N]`

- **개념적 타입**: 복잡한 타입의 구조를 잘 표현하기 위한 가상적인 타입 표현방식

`X a[N];`

C언어 타입



`X[N] a;`

개념적 타입

※ 주의
개념적 타입은 C언어의 문법적 표현이 아니므로
실제 프로그램 작성시 사용하면 안됨 → 구문오류

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

```
int main(void)
{
```

변수 a는
int 타입의 변수 3개로 구성된
복합변수

```
    int    a[3]; // ok: a[0] ~ a[2]
```

```
    int[3] b;    // syntax error
```

```
    return 0;
```

```
}
```

오히려 배열의 개념을
더 잘 전달하는 표현이다.

배열

■ 배열의 요소(element) 타입

`ET(X[N]) == X`

■ 배열의 크기

`X a[N];`

`sizeof(a) == N * sizeof(X)`

`Sizeof(a) == N * sizeof(ET(a))`

※ 주의
포인터의 크기는 word size이다. 배열의 크기와 다르다.

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

```
int main(void)
```

```
{
```

```
double a[3]={1.1,2.2,3.3};
```

```
double* p;
```

```
printf("%p %.2f %d\n",
```

```
&a[0], a[0], sizeof(a) );
```

```
p = &a[0];
```

```
printf("%p %.2f %d\n",
```

```
p, *p, sizeof(p) );
```

```
return 0;
```

```
}
```

입출력 결과

```
001CFE58 1.10 24
```

```
001CFE58 1.10 4
```

계속하려면 아무 키나 누르십시오 . . .

배열

■ 배열의 크기

```
X a[N];
```

```
sizeof(a) == N * sizeof(X)
```

■ 배열 요소의 개수

```
N == sizeof(a) / sizeof(X)
```

```
N == sizeof(a) / sizeof(a[0])
```

입출력 결과

24

8

3

계속하려면 아무 키나 누르십시오 . . .

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

```
int main(void)
```

```
{
```

```
    double a[]={1.1,2.2,3.3};
```

```
    int s;
```

```
    printf("%d\n", sizeof(a) );
```

```
    printf("%d\n", sizeof(a[0]) );
```

```
    s = sizeof(a) / sizeof(double);
```

```
    printf("%d\n", s );
```

```
    return 0;
```

```
}
```

 T(a) == double [3]

 T(a[0]) == double

포인터와 배열

■ 배열의 묵시적 타입 변환

$X[N] \rightarrow X^*$

- 배열은 변수이지만 복합체이므로 값(대표 값)을 정할 수 없다.
- 배열은 연산 중에 문제가 발생할 때만 묵시적 타입변환이 발생한다.
- 배열은 묵시적 타입 변환 후 배열의 값은 첫 번째 요소의 주소 값이 된다.

항목	변환 전	변환 후
타입	$X[N]$	X^*
값	복합 값	주소

```
#include <stdio.h>

int main(void)
{
    double a[3]={1.1,2.2,3.3};
    double* p;

    p = &a[0]; // double --> double*
    printf("%p %.2f %d\n",
           p, *p, sizeof(p) );

    p = a; // double[3] --> double*
    printf("%p %.2f %d\n",
           p, *p, sizeof(p) );
    return 0;
}
```

입출력 결과

```
0016FE50 1.10 4
0016FE50 1.10 4
계속하려면 아무 키나 누르십시오 . . .
```

포인터와 배열

- 배열의 묵시적 타입 변환
 - 배열은 복합 변수이므로 변수 대입 및 함수 인자 대입이 불가능하다.
 - 그러므로 **묵시적 타입 변환**이 발생한다.

입출력 결과

```
0016FE50 1.10 4
0016FE50 1.10 4
계속하려면 아무 키나 누르십시오 . . .
```

```
#include <stdio.h>

void f(double* p)
{
    printf("%p %.2f %d\n",
           p, *p, sizeof(p));
}

int main(void)
{
    double a[3]={1.1,2.2,3.3};
    double* p;

    p = a; // double[3] --> double*
    printf("%p %.2f %d\n",
           p, *p, sizeof(p) );

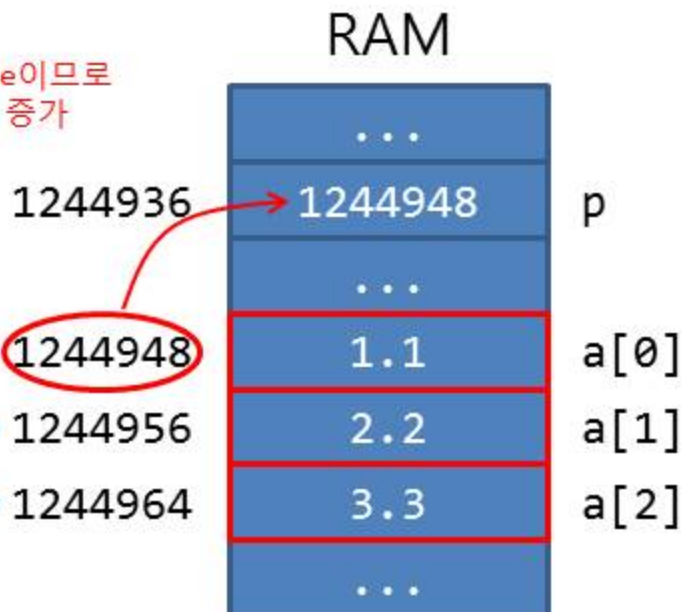
    f( a ); // double[3] --> double*
    return 0;
}
```

포인터와 배열

■ 포인터의 덧셈/뺄셈 연산

$p + i \rightarrow p + \text{sizeof}(\text{RT}(p)) * i$
 $p - i \rightarrow p - \text{sizeof}(\text{RT}(p)) * i$

RT가 double이므로
주소가 8씩 증가



※ 주의
 포인터의 곱셈과 나눗셈은 존재하지 않는다.

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

```
int main(void)
```

```
{
```

```
    double a[3]={1.1, 2.2, 3.3};
```

```
    double* p;
```

```
    p = &a[0];
```

```
    printf("%d %2f\n", p+0, *(p+0) );
```

```
    printf("%d %2f\n", p+1, *(p+1) );
```

```
    printf("%d %2f\n", p+2, *(p+2) );
```

```
    return 0;
```

```
}
```

입출력 결과

```
1244948 1.10
```

```
1244956 2.20
```

```
1244964 3.30
```

```
계속하려면 아무 키나 누르십시오 . . .
```

포인터와 배열

```
#include <stdio.h>

int main(void)
{
    double a[3] = {1.1,2.2,3.3};
    int i, s = sizeof(a)/sizeof(a[0]);

    for(i=0; i<s; ++i)
        printf("%.2f ", a[i] );
    printf("\n");
    return 0;
}
```

```
#include <stdio.h>

int main(void)
{
    double a[3] = {1.1,2.2,3.3};
    int i, s = sizeof(a)/sizeof(a[0]);
    double* p = a; // a → &a[0]

    for(i=0; i<s; ++i)
        printf("%.2f ", *(p+i) );
    printf("\n");
    return 0;
}
```


포인터와 배열

■ 포인터의 증감 연산

- 덧셈/뺄셈 연산을 사용
 - 내부적으로 곱셈 연산을 사용
 - 상대적으로 속도가 느림

```
p + i → p + sizeof(RT(p)) * i
p - i → p - sizeof(RT(p)) * i
```

• 증감 연산자의 사용

- 내부적으로 증감 연산만 사용
- 상대적으로 속도가 빠름

```
p++, ++p → p += sizeof(RT(p))
p--, --p → p -= sizeof(RT(p))
```

```
#include <stdio.h>

int main(void)
{
    double a[3]={1.1,2.2,3.3};
    double* p1 = &a[0];
    double* p2 = &a[0];

    printf("%p %p\n", p1+0, p2++ );
    printf("%p %p\n", p1+1, p2++ );
    printf("%p %p\n", p1+2, p2++ );
    return 0;
}
```

입출력 결과

```
0014F734 0014F734
0014F73C 0014F73C
0014F744 0014F744
계속하려면 아무 키나 누르십시오 . . .
```

포인터와 배열

- 포인터의 증감 연산을 사용한 배열 요소의 참조

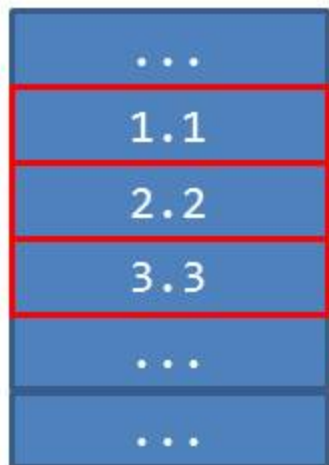
RAM

$\&a[0]+0 == 1008$

$\&a[0]+1 == 1016$

$\&a[0]+2 == 1024$

$\&a[0]+3 == 1032$



$a[0]$

$a[1]$

$a[2]$

$a[3]$

입출력 결과

1.10 2.20 3.30

계속하려면 아무 키나 누르십시오 . . .

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

```
int main(void)
```

```
{
```

```
    double a[3] = {1.1, 2.2, 3.3};
```

```
    double* p;
```

```
    p = &a[0];           // p == &a[0] + 0
```

```
    printf("%.2f ", *p); // *p == a[0]
```

```
    ++p;                // p == &a[0] + 1
```

```
    printf("%.2f ", *p); // *p == a[1]
```

```
    ++p;                // p == &a[0] + 2
```

```
    printf("%.2f ", *p); // *p == a[2]
```

```
    ++p;                // p == &a[0] + 3
```

```
    printf("\n");
```

```
    return 0;
```

```
}
```

$a[3]$ 은 실제 존재하지 않는 변수이지만,
포인터의 산술 연산 규칙에 따라
그 메모리 주소는 계산할 수 있다.

포인터와 배열

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

```
int main(void)
```

```
{
```

```
    double a[3] = {1.1,2.2,3.3};
```

```
    double* p;
```

```
    p = &a[0]; // p == &a[0] + 0
```

```
    printf("%.2f ", *p );
```

```
    ++p; // p == &a[0] + 1
```

```
    printf("%.2f ", *p );
```

```
    ++p; // p == &a[0] + 2
```

```
    printf("%.2f ", *p );
```

```
    ++p; // p == &a[0] + 3
```

```
    printf("\n");
```

```
    return 0;
```

```
}
```

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

```
int main(void)
```

```
{
```

```
    double a[3] = {1.1,2.2,3.3};
```

```
    double* p;
```

```
    for(p = &a[0]; p < &a[0]+3; ++p)  
        printf("%.2f ", *p );
```

```
    printf("\n");
```

```
    return 0;
```

```
}
```

포인터와 배열

인덱스 변수의 증감

```
#include <stdio.h>

int main(void)
{
    double a[3] = {1.1, 2.2, 3.3};
    int i, s = sizeof(a)/sizeof(a[0]);
    double* p = a;

    for(i=0; i<s; ++i)
        printf("%.2f ", *(p+i) );
    printf("\n");
    return 0;
}
```

입출력 결과

```
1.10 2.20 3.30
계속하려면 아무 키나 누르십시오 . . .
```

포인터 변수의 증감

```
#include <stdio.h>

int main(void)
{
    double a[3] = {1.1, 2.2, 3.3};
    int s = sizeof(a)/sizeof(a[0]);
    double* p;

    for(p=a; p < a+s; ++p)
        printf("%.2f ", *p );
    printf("\n");
    return 0;
}
```

입출력 결과

```
1.10 2.20 3.30
계속하려면 아무 키나 누르십시오 . . .
```

포인터와 배열

■ 포인터와 정수의 덧셈/뺄셈

```
T( p + i ) == X*
T( p - i ) == X*
T(p) == X*, T(i) == int
```

```
T( p - q ) == int
T(p) == X*, T(q) == X*
```

```
T( p + q ) == Error
T(p) == X*, T(q) == X*
```

입출력 결과

```
0029F720 --> 1.100000
0029F730 --> 3.300000
0029F730 - 0029F720 == 2
계속하려면 아무 키나 누르십시오 . . .
```

```
#include <stdio.h>

int main(void)
{
    double a[3] = {1.1, 2.2, 3.3};
    double* p = &a[0];
    double* q;
    int i = 2;

    q = p + i; // T(p+i) == double*
    printf("%p --> %f\n", p, *p );
    printf("%p --> %f\n", q, *q );

    i = q - p; // ok: T(q-p) == int
    printf("%p - %p == %d\n", q, p, i );

    // i = p + q; // error
    return 0;
}
```

포인터와 배열

■ 배열 연산자의 의미

`p[i] == *(p + i)`

- 포인터를 배열처럼 활용 가능

■ 배열 연산자

연산자	사용 방법	결과
<code>[]</code>	주소[인덱스]	변수

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

```
int main(void)
```

```
{
```

```
    double a[3]={1.1,2.2,3.3};
```

```
    double* p;
```

```
    p = &a[0]; // double --> double*
```

```
    printf("%.2f  %.2f\n", p[0], *(p+0) );
```

```
    printf("%.2f  %.2f\n", p[1], *(p+1) );
```

```
    printf("%.2f  %.2f\n", p[2], *(p+2) );
```

```
    return 0;
```

```
}
```

입출력 결과

```
1.10  1.10
```

```
2.20  2.20
```

```
3.30  3.30
```

```
계속하려면 아무 키나 누르십시오 . . .
```

포인터 배열

개별 포인터

```
#include <stdio.h>

int main(void)
{
    int x=11, y=22, z=33;
    int* a0;
    int* a1;
    int* a2;

    a0 = &x;
    a1 = &y;
    a2 = &z;

    printf("%d\n", *a0);
    printf("%d\n", *a1);
    printf("%d\n", *a2);
    return 0;
}
```

포인터 배열

```
#include <stdio.h>

int main(void)
{
    int x=11, y=22, z=33;
    int* a[3];

    a[0] = &x;
    a[1] = &y;
    a[2] = &z;

    printf("%d\n", *a[0]);
    printf("%d\n", *a[1]);
    printf("%d\n", *a[2]);
    return 0;
}
```

포인터 배열

- 포인터 배열
 - 포인터들로 구성된 배열

$X^* a[N];$

$ET(a) == X^*$
 $T(a) == X^*[N]$

```
#include <stdio.h>

int main(void)
{
    int x=11, y=22, z=33;
    int* a[3] = { &x, &y, &z };
    // 요소타입이 int*

    printf("%d\n", *a[0]);
    printf("%d\n", *a[1]);
    printf("%d\n", *a[2]);
    return 0;
}
```

입출력 결과

```
11
22
33
계속하려면 아무 키나 누르십시오 . . .
```

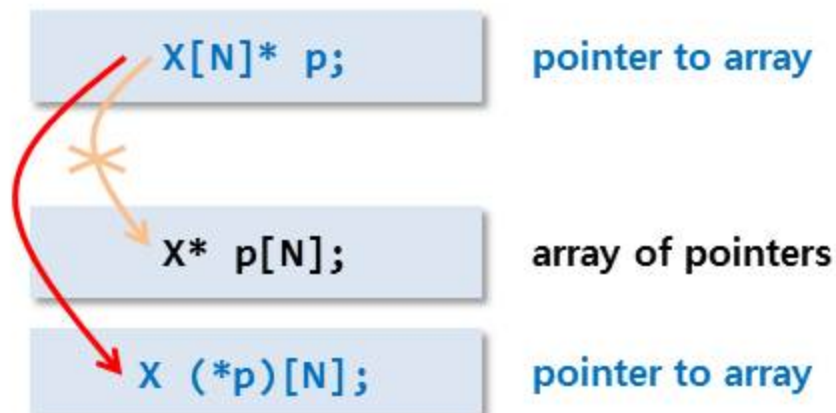

배열 포인터

■ 배열의 포인터 타입

```
X a[N];
T(a) == X[N];
```

```
PT(a) == X[N]*
T(&a) == X[N]*
```

■ 개념적 타입의 전환



```
#include <stdio.h>

int main(void)
{
    int a[3];
    int (*p)[3];

    p = &a; // T(&a) == T(p) == int[3]*

    (*p)[0] = 11;
    (*p)[1] = 22;
    (*p)[2] = 33;
    return 0;
}
```


복합체의 포인터

■ 일반 자료형의 포인터

`PT(X) == X*`

■ 구조체의 포인터

`PT(struct X) == struct X*`

■ 배열의 포인터

`PT(X[N]) == X[N]*`

```
#include <stdio.h>

struct stud {
    int id;
    char name;
    double grade;
};

int main(void)
{
    struct stud s;
    int a[3];
    struct stud* ps = &s; // struct stud*
    int (*pa)[3] = &a;    // int[3]*

    (*ps).id=10;
    (*pa)[0]=1;
    return 0;
}
```

배열 요소 포인터

- 배열 요소 포인터(pointer to array element)

$$\begin{aligned} \text{PT}(\text{ET}(\text{X}[\text{N}])) &== \text{X}^* \\ \text{EPT}(\text{X}[\text{N}]) &== \text{X}^* \end{aligned}$$

RT가 double이므로
주소가 8씩 증가

...	...	
1244936	1244948	p
	...	
p+0==1244948	1.1	a[0]
p+1==1244956	2.2	a[1]
p+2==1244964	3.3	a[2]
	...	

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

```
int main(void)
{
```

```
    double a[3]={1.1,2.2,3.3};
```

```
    double* p;
```

```
    p = &a[0];
```

p는 배열 a의 요소를
가리키고 있으므로
배열 요소 포인터이다.

```
    printf("%.2f ", *(p+0));
    printf("%.2f ", *(p+1));
    printf("%.2f ", *(p+2));
    printf("\n");
```

```
    printf("%.2f ", p[0]);
    printf("%.2f ", p[1]);
    printf("%.2f ", p[2]);
    printf("\n");
```

동일한
표현

$$p[i] == *(p + i)$$

```
    return 0;
```

```
}
```

배열 요소 포인터

- 배열 요소 포인터의 장점
 - 배열 요소 포인터를 사용하면 배열처럼 요소 참조가 가능
- 배열 포인터의 타입

PT(X[N]) == X[N]*



EPT(X[N]) == X*

```
#include <stdio.h>

int main(void)
{
    double a[3]={1.1,2.2,3.3};
    double* p;

    p = a; // p = &a[0];

    printf("%.2f ", *(p+0) );
    printf("%.2f ", *(p+1) );
    printf("%.2f ", *(p+2) );
    printf("\n");

    printf("%.2f ", p[0] );
    printf("%.2f ", p[1] );
    printf("%.2f ", p[2] );
    printf("\n");

    printf("%.2f ", a[0] );
    printf("%.2f ", a[1] );
    printf("%.2f ", a[2] );
    printf("\n");
    return 0;
}
```

포인터를
배열처럼
사용가능

배열을 위한 포인터

배열 포인터

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

```
int main(void)
```

```
{
```

```
    double a[3];
```

```
    double (*p)[3] = &a;
```

```
    (*p)[0] = 1.1;
```

```
    (*p)[1] = 2.2;
```

```
    (*p)[2] = 3.3;
```

```
    return 0;
```

```
}
```

$T(&a) == \text{double}[3]^*$

배열 포인터 보다
배열 요소 포인터가
더 편리하다.

배열 요소 포인터

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

```
int main(void)
```

```
{
```

```
    double a[3];
```

```
    double *p = a;
```

```
    p[0] = 1.1;
```

```
    p[1] = 2.2;
```

```
    p[2] = 3.3;
```

```
    return 0;
```

```
}
```

$T(a) == \text{double}[3]$
 $\text{double}[3] \rightarrow \text{double}^*$

배열을 위한 포인터

배열 포인터

```
#include <stdio.h>

void print_array( double (*p)[3] )
{
    int i;

    for(i=0;i<3;++i)
        printf("%g ", (*p)[i] );
    printf("\n");
}

int main(void)
{
    double a[3] = {1.1, 2.2, 3.3};

    print_array( &a );
    return 0;
}
```

$T(&a) == \text{double}[3]^*$

배열 요소 포인터

```
#include <stdio.h>

void print_array( double* p )
{
    int i;

    for(i=0;i<3;++i)
        printf("%g ", p[i] );
    printf("\n");
}

int main(void)
{
    double a[3] = {1.1, 2.2, 3.3};

    print_array( a );
    return 0;
}
```

배열 포인터 보다
배열 요소 포인터가
더 편리하다.

$T(a) == \text{double}[3]$
 $\text{double}[3] \rightarrow \text{double}^*$

포인터의 참조대상

■ 포인터와 배열 포인터

PT(T) == T*
EPT(T[N]) == T*

- T의 포인터 타입과 T[N]의 배열 요소 타입이 동일하다.
- 포인터 타입 T*의 원타입을 알기 위해서는 T*의 목적을 파악하는 것이 중요하다.

입출력 결과

```
002CF7F8 1.10 4
002CF7E8 4.40 4
계속하려면 아무 키나 누르십시오 . . .
```

```
#include <stdio.h>

int main(void)
{
    double a[3]={1.1,2.2,3.3};
    double b=4.4;
    double* p;

    p = a; // PT( double[3] ) == double*
    printf("%p %.2f %d\n",
           p, *p, sizeof(p) );

    p = &b; // PT( double ) == double*
    printf("%p %.2f %d\n",
           p, *p, sizeof(p) );
    return 0;
}
```

문자열 상수(string constant)

- 문자열 상수(= char 배열)의 포인터
 - char 배열의 요소 포인터를 사용하여 문자열의 주소를 저장

```
#include <stdio.h>

int main(void)
{
    char* p = "abc"; // char[4] --> char*
    char* q;

    q = "abc";       // char[4] --> char*
    return 0;
}
```

Diagram: A red arrow points from the string literal "abc" in the line `char p = "abc";` to the text `T("abc") == char[4]`. Another red arrow points from the variable `p` in the same line to the text `T(p) == char*`.*

문자열 상수(string constant)

- 문자열 배열 초기화
 - 문자열 상수처럼 보이지만 **문자열 상수가 아니고 initializer**이다.
- 문자열 상수의 주소 값 저장
 - char 배열의 요소 포인터가 사용될 때 **문자열 상수**가 된다.

```
#include <stdio.h>

int main(void)
{
    char a[] = "abc";
    char a[] = {'a', 'b', 'c', '\0'};
    char* p = "abc";
    return 0;
}
```

Diagram annotations:

- Red arrow from `char a[] = "abc";` to `T(a) == char[4]`
- Red arrow from `char a[] = {'a', 'b', 'c', '\0'};` to `initializer`
- Red arrow from `char* p = "abc";` to `string constant`
- Red arrow from `char* p = "abc";` to `T(p) == char*`

Left side annotation: **같은 의미** (Same meaning) with a bracket pointing to both array initialization lines.

문자열 상수(string constant)

■ 문자열 변경

- 배열 요소 포인터

- 배열 요소 포인터는 문자열 상수의 주소를 저장하므로 주소 값의 변경이 가능하다.


- char 배열

- 배열은 묵시적 타입 변환으로 주소 값으로 변할 수 있지만, 그 배열의 첫 번째 요소의 주소 값을 변경할 수는 없다.

```
#include <stdio.h>

int main(void)
{
    char* p = "abc";
    char a[] = "abc";

    p = "efg";    // ok: char[4] --> char*
    a = "efg";    // syntax error
    return 0;
}
```

 a - - > &a[0]
a[0]의 주소 값에
문자 상수의 주소 값을
저장하지 못함

복합 변수의 값 복사

■ 구조체의 복사

- 전통 C언어에서는 구조체의 대입 연산을 지원하지 않는다.
- C++언어에서는 언어의 특성상 구조체 복사가 꼭 필요하므로 최근 컴파일러들은 이를 허용하고 있다.
- 하지만, 구조체의 크기가 커질수록 구조체 복사는 시간이 많이 걸린다. 이는 프로그램 속도 저하의 원인이 되므로 남용해서는 안 된다.
- 본 강의에서도 구조체 복사는 추천하지 않는다.

```
#include <stdio.h>

struct stud {
    int id;
    char name;
    double grade;
};

int main(void)
{
    struct stud s1={25,'A',3.5};
    struct stud s2;

    s2=s1; // undesirable: error in old C.
    return 0;
}
```

복합 변수의 값 복사

■ 배열의 복사

- C언어에서는 배열의 대입연산을 지원하지 않는다. (이는 C++에서도 마찬가지다.)
- 대입 연산자를 사용할 경우 배열의 묵시적 타입변환이 일어난다. 이때 대입연산자의 좌측 피연산자가 값(주소)이므로 오류가 발생한다.

`b = a;` → `&b[0] = &a[0];`
주소값 = 주소값

```
#include <stdio.h>

int main(void)
{
    int a[10]={1,2,3,4,5,6,7,8,9,10};
    int b[10];

    b=a; // error: &b[0]=&a[0]
    return 0;
}
```

복합 변수의 관계연산

- 구조체의 관계연산
 - 구조체에 대해서 관계연산자는 정의되어 있지 않다.
 - 이 경우 **구문오류**가 발생한다.

```
#include <stdio.h>

struct point {
    double x, y;
};

int main(void)
{
    struct point a = {1.0, 2.0};
    struct point b = {2.0, 3.0};

    if( a < b ) // syntax error
        printf("less then\n");
    return 0;
}
```

→ 구조체의 대소 비교는 불가능하다.

복합 변수의 관계연산


■ 배열의 관계연산

- 배열은 관계연산이 불가능하다
- 그러므로 배열의 **묵시적 타입** 변환이 일어난다.
- 결과적으로 배열의 관계연산은 **첫 번째 요소들의 주소를 비교**하는 것에 불과하기 때문에 **의도하는 바와 다르게 동작**할 수 있으며 이에 따라 **논리오류**를 발생시킬 수 있다.

```
#include <stdio.h>

int main(void)
{
    int a[3]={1,2,3};
    int b[3]={2,3,4};

    if( a < b ) // &a[0] < &b[0]
        printf("less then\n");
    return 0;
}
```

 **a[0], b[0]의 주소가 어떤 값일지 예상하지 못하므로 이 경우 관계연산자의 결과 값을 예상할 수 없다.**

함수

■ 구조체의 전달

- 전통 C언어에서는 구조체의 대입 연산을 지원하지 않는다.
- 그러므로 함수 인자 전달로 일반적으로 구조체 복사를 사용하지 않는다.
- 구조체의 경우 주소를 전달하는 것이 효과적이다.
- 일반적으로 구조체 전달을 위한 함수의 인자 타입으로 **구조체 포인터**를 사용한다.

입출력 결과

11 b 4.140000

계속하려면 아무 키나 누르십시오 . . .

```
#include <stdio.h>

struct stud {
    int id;
    char name;
    double grade;
};

void funct( )
{
    p->id = 11;
    p->name = 'b';
    p->grade = 4.14;
}

int main(void)
{
    struct stud s = {10, 'a', 3.14};

    funct( &s );
    printf("%d %c %f\n",
        s.id, s.name, s.grade);
    return 0;
}
```


함수

■ 배열의 전달

- C언어에서는 배열의 대입연산을 지원하지 않는다.
- 그러므로 함수 인자 전달로 배열 복사의 사용이 불가능하다.
- 배열의 경우 주소를 전달하는 것이 유일한 방법이다.
- 일반적으로 배열 전달을 위한 함수의 인자 타입은 **배열 요소 포인터**를 사용한다.

입출력 결과

```
1 2 3
계속하려면 아무 키나 누르십시오 . . .
```

```
#include <stdio.h>

void funct( )
{
    p[0] = 1;
    p[1] = 2;
    p[2] = 3;
}

int main(void)
{
    int a[3] = {0, 0, 0};
    int s = sizeof(a)/sizeof(a[0]);
    int i;

    funct( a );    // int[3] --> int*
    for(i=0;i<s;++i)
        printf("%d ",a[i]);
    printf("\n");
    return 0;
}
```

함수

- 배열의 전달을 위한 인자의 표현 방식
 - 배열(array)
 - 생략형 배열(omitted array)
 - 포인터(pointer)
- 실제 처리 방식
 - 포인터(pointer)

입출력 결과

```
main(): 12
f1(): 4
f2(): 4
f3(): 4
계속하려면 아무 키나 누르십시오 . . .
```

```
#include <stdio.h>

void f1( int a[3] ) // array
{
    printf("f1(): %d\n", sizeof(a) );
}

void f2( int a[] ) // omitted array
{
    printf("f2(): %d\n", sizeof(a) );
}

void f3( int* a ) // pointer
{
    printf("f3(): %d\n", sizeof(a) );
}

int main(void)
{
    int a[3] = {1, 2, 3};

    printf("main(): %d\n", sizeof(a) );
    f1( a );
    f2( a );
    f3( a );
    return 0;
}
```

함수

- 배열의 길이 전달
 - 배열 인자만으로는 배열의 길이가 전달되지 않는다.
- 방법
 - 배열의 길이 전달을 위한 추가 인자 사용
 - 외부 변수나 매크로 사용

입출력 결과

```
main(): size=3
f(): size=1
1
계속하려면 아무 키나 누르십시오 . . .
```

```
#include <stdio.h>

void f(int a[3])
{
    int i;
    int s=sizeof(a)/sizeof(a[0]); // meaningless

    printf("f(): size=%d\n",s);
    for(i=0;i<s;++i)
        printf("%d\n",a[i]);
}

int main(void)
{
    int a[3] = {1,2,3};
    int s=sizeof(a)/sizeof(a[0]); // meaningful

    printf("main(): size=%d\n",s);
    f(a);
    return 0;
}
```

함수

외부 변수 사용한 배열 길이 전달

```
#include <stdio.h>

int size;

void f(int a[3]) // int a[3] --> int* a
{
    int i;

    printf("f(): size=%d\n", size);
    for(i=0; i<size; ++i)
        printf("%d ", a[i]);
    printf("\n");
}

int main(void)
{
    int a[3] = {1, 2, 3};

    size = sizeof(a)/sizeof(a[0]);
    printf("main(): size=%d\n", size);
    f(a);
    return 0;
}
```

매크로 사용한 배열 길이 전달

```
#include <stdio.h>

#define SIZE 3

void f(int a[SIZE]) // int a[3] --> int* a
{
    int i;

    printf("f(): size=%d\n", SIZE);
    for(i=0; i<SIZE; ++i)
        printf("%d ", a[i]);
    printf("\n");
}

int main(void)
{
    int a[SIZE] = {1, 2, 3};

    printf("main(): size=%d\n", SIZE);
    f(a);
    return 0;
}
```

함수

인자를 사용한 길이 전달

```
#include <stdio.h>

void f(int a[3], int size)
{
    int i;

    printf("f(): size=%d\n", size);
    for(i=0; i<size; ++i)
        printf("%d ",a[i]);
    printf("\n");
}

int main(void)
{
    int a[3] = {1, 2, 3};
    int size = sizeof(a)/sizeof(a[0]);

    printf("main(): size=%d\n", size);
    f(a, size);
    return 0;
}
```

특정 값을 사용한 길이 전달

```
#include <stdio.h>

void f(int a[3])
{
    int i;

    for(i=0; a[i]!=-1; ++i)
        printf("%d ",a[i]);
    printf("\n");
}

int main(void)
{
    int a[4] = {1, 2, 3, -1};

    f(a);
    return 0;
}
```

함수

- 문자열의 길이 전달
 - 특정 값(NULL)을 사용한 크기 전달
 - 주의
 - 배열의 길이: 4
 - 문자열의 길이: 3

입출력 결과

abc

계속하려면 아무 키나 누르십시오 . . .

```
#include <stdio.h>

void f(char a[])
{
    int i;

    for(i=0; a[i]!='\0'; ++i)
        printf("%c", a[i]);
    printf("\n");
}

int main(void)
{
    char a[4] = {'a','b','c','\0'};

    f(a);
    return 0;
}
```