

C 언어 기초 II

작성자 : 홍효상
이메일 : hyomee@naver.com
소스 : https://github.com/hyomee/c_basic

Content

I. C 언어 기초 II

1. 배열
2. 포인터
3. C언어 고급 기본

Content

I. 배열

1. 배열
2. 문자열
3. 2차원 배열
4. CodeUp 예제 풀이

“ 데이터(정보) 중 같은 자료형 정보를 그룹으로 표현 ”

배열 의미

- 자료형 또는 컴퓨터공학에서 사용하는 자료구조의 하나
- 순서대로 번호가 붙은 원소들이 연속적인 형태로 구성된 구조
- 각 원소에 붙은 번호를 흔히 첨자(인덱스, index)
- 같은 타입의 변수들로 이루어진 유한 집합



- Int 형 n+1 개의 배열 -> int형 변수 n개 선언

배열 선언, 사용

- 배열 선언: 자료형 배열이름[크기]

- 배열 선언 : short price[5] ;
- 배열 초기화 : 배열이름[Index] = 값
- price[1] = 10

```
for ( index = 0; index < 배열크기 - 1 ; index++) {  
    ....  
}
```

- 배열 선언 및 : 자료형 배열이름[크기] = { 값, 값 ... 값 }

- 배열 선언 및 초기화 : short price[5] = { 1, 2, 3, 4, 5 }
- 배열의 모든 요소 동일 값 : short price[5] = { 1, } : { 1, 0, 0, 0, 0 } 와 동일
- 배열 크기 생략 : short price[] = { 1, 2, 3, 4, 5 } => short price[5] = { 1, 2, 3, 4, 5 } 와 동일

배열 선언, 사용

- 배열의 길이: 배열크기 / 자료형 => sizeof(배열명) / sizeof(자료형)
- 배열 선언 만 하면 각 요소는 쓰레기 값이 들어 있다.
- 배열 요소에 값을 저장 하면 해당 요소 만 값이 변경 됨
- 배열 요소보다 큰 값을 저장 하면 크기와 길이는 변경 하지 않는다. 즉 선언된 길이 보다 크면 모두 쓰레기 값이다.

```
#include <stdio.h>

void main() {
    int index ;
    int englishScore[5];
    int englishScoreLegnth = sizeof(englishScore)/sizeof(int);
    englishScore[1] = 10;
    englishScore[4] = 200;
    printf("크기 : %d , 길이 : %d\n", sizeof(englishScore), englishScoreLegnth );

    for ( index = 0; index < englishScoreLegnth ; index++) {
        printf("영어 점수 : %d index : %d\n", index, englishScore[index]);
    }
    printf("영어 점수 : %d index : %d\n", 5, englishScore[5]);

    englishScore[5] = 500;
    printf("크기 : %d , 길이 : %d\n", sizeof(englishScore),
    sizeof(englishScore)/sizeof(int));
    printf("영어 점수 : %d index : %d\n", 5, englishScore[5]);
}
```

• 초기값

```
▼ englishScore: [5]
[0]: 8
[1]: 0
[2]: 39
[3]: 0
[4]: 13062000
englishScoreLegnth: 5
```

• 중간값

```
▼ englishScore: [5]
[0]: 8
[1]: 10
[2]: 39
[3]: 0
[4]: 200
englishScoreLegnth: 5
```

• 결과

```
크기 : 20 , 길이 : 5
영어 점수 : 0 index : 8
영어 점수 : 1 index : 10
영어 점수 : 2 index : 14
영어 점수 : 3 index : 0
영어 점수 : 4 index : 200
영어 점수 : 5 index : 0
크기 : 20 , 길이 : 5
영어 점수 : 5 index : 500
```

1. 배열

1. 배열 1-1. 배열

연속적인 숫자 합

```
#include <stdio.h>
void main() {
    int index , sum = 0 ;
    char score[5] = {1,2,3,4,5};
    int scoreLegnth = sizeof(score)/sizeof(char);

    for ( index = 0; index < scoreLegnth ; index++) {
        sum = sum + score[index];
    }
    printf("sum : %d \n", sum);
}
```

```
#include <stdio.h>
void main() {
    int index , sum = 0 ;
    char score[5] ;
    printf("숫자를 입력 하세요 : \n");
    scanf("%d", &score[0]);
    scanf("%d", &score[1]);
    scanf("%d", &score[2]);
    scanf("%d", &score[3]);
    scanf("%d", &score[4]);

    int scoreLegnth = sizeof(score)/sizeof(char);
    for ( index = 0; index < scoreLegnth ; index++) {
        sum = sum + score[index];
    }
    printf("sum : %d \n", sum);
}
```

구구단

```
#include <stdio.h>

/**
 * @brief 임의의 숫자 5개를 입력 받아서 합 구하기
 *
 */
void main() {
    int index , sum = 0, num;
    char count[9] = {1,2,3,4,5,6,7,8,9};
    char result[9];
    printf("몇단을 출력 할까요 ? : ");
    scanf("%d", &num);
    printf("%d단===== \n", num );
    int length = sizeof(count)/sizeof(char);

    for ( index = 0; index < length ; index++) {
        result[index] = num * count[index];
        printf("%d * %d = %d \n", num, count[index],
result[index]);
    }

    for ( index = 0; index < length ; index++) {
        sum = sum + result[index];
    }
    printf("%d단의 전체 합 : %d \n", num, sum);
}
```

“ 문자들의 집합 ”

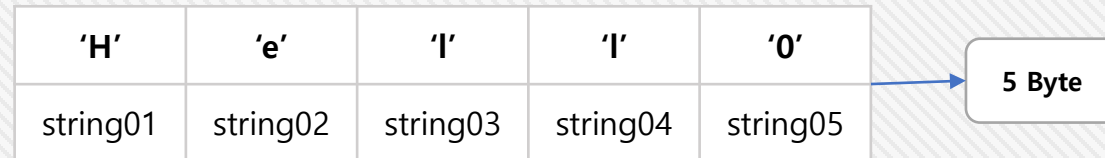
문자열

- 문자들의 열
- A는 문자, ABCD는 문자열
- 배열 or Pointer 로 표시
- 배열로 저장 시: 문자 정보 + 문자 개수 -> 문자정보 + null문자(0)
- 표기법

```
char 배열이름[크기] = { 값, 값 ... 값 };  
char 배열이름[] = { 값, 값 ... , 0 };  
char 배열이름[] = "값";
```

문자형

```
char string01 = 'H';  
char string02 = 'e';  
char string03 = 'l';  
char string04 = 'l';  
char string05 = 'o';
```



배열

```
char string[] = {'H', 'e', 'l', 'l', 'o', 0};
```



마지막 null 처리
하지 않음

```
char string[] = {'H', 'e', 'l', 'l', 'o'};  
printf("%s, %d", string, sizeof(string));
```

Helloo?, 5

같은 의미

```
char string[] = {'H', 'e', 'l', 'l', 'o', 0};  
char string[6] = {'H', 'e', 'l', 'l', 'o'};  
char string[] = "Hello";
```

문자열 저장

- char str[6] = "Hello";

배열	str[0]	str[1]	str[2]	str[3]	str[4]		
pointer	*str	*(str+1)	*(str+2)	*(str+3)	*(str+4)		
메모리공간	'H'	'e'	'l'	'l'	'o'		
주소값	1001	1002	1003	1004	1005		
pointer연산	str	str+1	str+2	str+3	str+4		

문자 상수 / 문자열 상수

- 문자 하나: 'A' (작은 따옴표)
- 문자 집합: "ABCD" (큰 따옴표)

배열 사용시 속도 향상

- 문자 정보와 문자 개수를 각각 선언 사용
- 어떤 방식으로 하여도 결과는 같다 -> 속도 우선

```
#include <stdio.h>

void main() {
    int index ;
    char string[5] = {'H', 'e', 'l', 'l', 'o'};
    char stirngLength = 5;

    for ( index = 0; index <= stirngLength; index++ ){
        printf("%c", string[index]);
    }
}
```

```
#include <stdio.h>

void main() {
    int index ;
    char string[6] = {'H', 'e', 'l', 'l', 'o'};

    for ( index = 0; index <= 6; index++ ){
        printf("%c", string[index]);
    }
}
```

```
#include <stdio.h>

void main() {
    int index ;
    char string[] = "Hello";
    char stringLength = sizeof(string);

    for ( index = 0; index < stringLength; index++ ){
        printf("%c", string[index]);
    }
}
```

문자열 에서 문자 정보 길이

- 배열로 문자열 표현 시 배열의 길이가 문자정보의 길이가 아니다.

```
#include <stdio.h>

int getStringLength(char data[]) {
    int count = 0;
    while(data[count]) {
        count++;
    }
    return count;
}
```

```
void main() {
    int index ;
    char string[] = "Hello";
    int stringLength = getStringLength(string);

    for ( index = 0; index <= stringLength; index++ ){
        printf("%c", string[index]);
    }
}
```

문자열 내장 함수

- string.h에 정의 됨
- strlen(), strcpy(), strcat() -> <https://www.csse.uwa.edu.au/programming/ansic-library.html>

➤ 문자열 배열 에서 문자정보 길이 구함 : `size_t strlen(const char* cs);`

```
#include <stdio.h>
#include <string.h>
void main() {
    char data[] = "Hello";
    int dataLength;
    dataLength = strlen(data);
    printf("data : %s \ndata legnth : %d\n", data, dataLength);
}
```

```
data : Hello
data legnth : 5
```

➤ 문자열 복사 : `char* strcpy(char* s, const char* ct);`

```
#include <stdio.h>
#include <string.h>
void main() {
    char sourceData[] = "Hello";
    int sourceDataLength = sizeof(sourceData);
    char targetData[sourceDataLength];
    strcpy(targetData, sourceData);
    printf("sourceData : %s \ntargetData : %s\n", sourceData, targetData);
}
```

```
sourceData : Hello
targetData : Hello
```

➤ 문자열 합치기 : `char* strcat(char* s, const char* ct);`

```
#include <stdio.h>
#include <string.h>
void main() {
    char source[6] = "Hello";
    char target[11] = "World";
    strcat(target, source);
    printf("source : %s \nsourceSize : %d\n", source, sizeof(source));
    printf("target : %s \ntargetSize : %d \ntargetLength : %d\n", target,
    sizeof(target), strlen(target));
}
```

```
source : Hello
sourceSize : 6
target : WorldHello
targetSize : 11
targetLength : 10
```

➤ 문자열 일부 복사 : `char* strncpy(char* s, const char* ct);`

```
#include <stdio.h>
#include <string.h>
void main() {
    char source[12] = "Hello World";
    char target[12] = "xxxxx World";
    strncpy(target, source, 5);
    printf("source : %s \ntarget : %s\n", source, target);
}
```

```
source : Hello World
target : Hello World
```

배열 반환

- C언어에서는 배열을 그대로 넘길 수 없고, 포인터로만 사용
- 1차원 배열은 포인터 연산(+n)과 인덱스[n]이 일치 하므로 간단 하나 2차원 이상의 배열은 배열 구조에 맞추어야 하므로 변환 과정이 필요 함
- 함수를 반환 하기 위한 변수는 static으로 선언 한다. (지역 변수는 언제나 사라질 수 있음)

➤ 정수형 배열

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

int* getNmberArray();
void printNumberArray();

char* getCharArray();
void printCharArray();
void printCharArrayStrcpy();

int i;

void main() {

    printNumberArray();
    printCharArray();
    printCharArrayStrcpy();

}
```

```
void printNumberArray() {
    printf("==== int Array =====\n");
    int* nums = getNmberArray();
    for( i=0; i<6; i++) {
        printf("%d", nums[i]);
    }
}

int* getNmberArray() {
    static int arr[6] = {1,2,3,4,5,6};
    return arr;
}
```

➤ 문자열 배열

```
void printCharArray() {
    printf("\n==== int Char =====\n");
    char* chars = getCharArray();
    for( i=0; i<6; i++) {
        printf("%c", chars[i]);
    }
}

void printCharArrayStrcpy() {
    printf("\n==== int Char 02 =====\n");
    char* charArray = getCharArray();
    strcpy(charArray, getCharArray());
    printf("%s",charArray);
}

char* getCharArray() {
    static char chars[6] = {'a', 'b', 'c', 'd'};
    return chars;
}
```

1. 2차원 배열

1. 배열 1-3. 2차원 배열

2차원 배열

- 행과 열을 사용 해서 저장

행	열	0	1	2
		0,1	0,1	0,2
1		1,0	1,1	1,2
2		2,0	2,1	2,2

	C : 0	C++ : 1	JAVA : 2
홍길동 : 0	80 : 0,0	90 : 0,1	100 : 0,2
바둑이 : 1	70 : 1,0	50 : 1,1	40 : 1,2
김영이 : 2	90 : 2,0	80 : 2,1	100 : 2,2
정정이 : 3	60 : 3,0	80 : 3,1	80 : 3,2

```
#include <stdio.h>

void main() {
    char score[12] = {80, 90, 100, 70, 50, 40, 90, 80, 100, 60, 80, 80};
    int index, x, y;

    for ( index = 0; index < 12 ; index ++ ) {
        x = index / 3 ;
        if ( x == 0 ) {
            printf("C : %d \n", score[index]);
        } else if ( x == 1 ) {
            printf("C++ : %d \n", score[index]);
        } else if ( x == 2 ) {
            printf("JAVA : %d \n", score[index]);
        }
    }
}
```

```
#include <stdio.h>

void main() {
    char score[12] = {80, 90, 100, 70, 50, 40, 90, 80, 100, 60, 80, 80};
    int index, x, y;

    for ( index = 0; index < 12 ; index ++ ) {
        y = index / 4 ;
        if ( y == 0 ) {
            printf("홍길동 : %d \n", score[index]);
        } else if ( y == 1 ) {
            printf("바둑이 : %d \n", score[index]);
        } else if ( y == 2 ) {
            printf("김영이 : %d \n", score[index]);
        } else if ( y == 3 ) {
            printf("정정이 : %d \n", score[index]);
        }
    }
}
```

1. 2차원 배열

1. 배열 1-3. 2차원 배열

2차원 배열

- 행과 열을 사용 해서 저장

```
#include <stdio.h>

void main() {
    char score[4][3] = {{80, 90, 100}, {70,50,40}, {90,80,100}, {60,80,80}};
    int stud[4] ;
    char studName[4] = {'A', 'B', 'C', 'D'};
    int c, r;
    int cSum = 0, cplusSum = 0, javaSum = 0 ;
```

// 언어별

```
for ( r = 0; r < 4 ; r ++ ) {
    int sum = 0;
    for ( c = 0; c < 3 ; c ++ ) {
        sum += score[r][c];
        if (c == 0) {
            cSum += score[r][c];
        } else if (c == 1) {
            cplusSum += score[r][c];
        } else if (c == 2) {
            javaSum += score[r][c];
        }
    }
    stud[r] = sum;
}
```

```
printf("C : %d \n", cSum);
printf("C++ : %d \n", cplusSum);
printf("JAVA : %d \n", javaSum);
for ( r = 0; r < 4 ; r ++ ) {
    printf("%c : %d \n", studName[r], stud[r]);
}

}
```

```
C : 300
C++ : 300
JAVA : 320
A : 270
B : 160
C : 270
D : 220
```

2차원 배열

- 행과 열을 사용 해서 저장

```
#include <stdio.h>

void main() {
    char score[4][3] = {{80, 90, 100}, {70,50,40}, {90,80,100}, {60,80,80}};
    int stud[4] ;
    int scoreSum[3] = { 0,0,0};
    char scoreSumName[3][10] = { "C", "C++", "JAVA"};
    char studName[4][30] = {"홍길동", "바둑이", "김영이", "정정이"};
    int c, r;

    for ( r = 0; r < 4 ; r ++ ) {
        int sum = 0;
        for ( c = 0; c < 3 ; c ++ ) {
            sum += score[r][c];
            scoreSum[c] += score[r][c];
        }
        stud[r] = sum;
    }

    for ( c = 0; c < 3 ; c ++ ) {
        printf("%s : %d \n", scoreSumName[c], scoreSum[c]);
    }
    for ( r = 0; r < 4 ; r ++ ) {
        printf("%s : %d \n", studName[r], stud[r]);
    }
}
```

```
C : 300
C++ : 300
JAVA : 320
홍길동 : 270
바둑이 : 160
김영이 : 270
정정이 : 220
```

Content

II. 포인터

1. 메모리 관리
2. 포인터 기초
3. 배열과 포인터
4. 변환 (자료형 반환)

“ Heap를 사용 하는 메모리는 개발자가 관리 하여야 하므로 할당 및 해제를 해야 함”

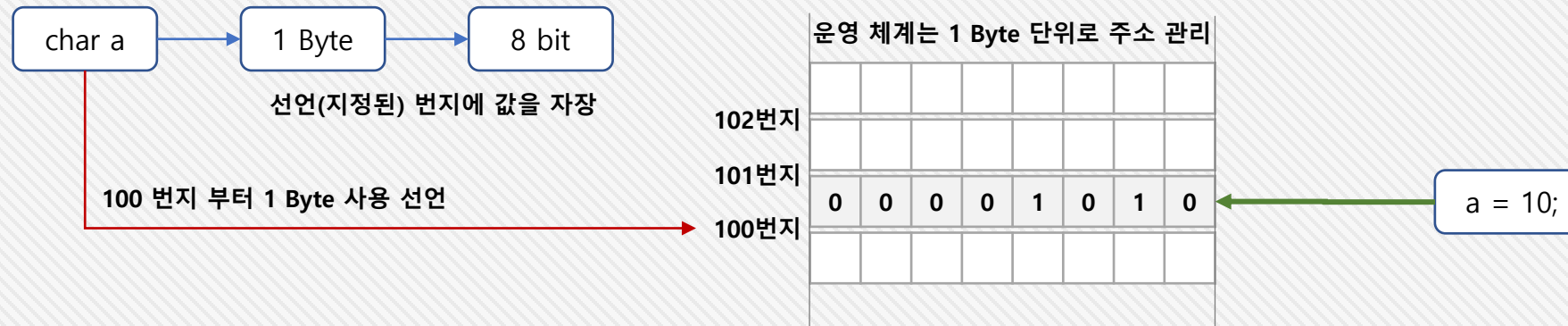
메모리 종류

분 류		특 징
Stack		자동 변수 지역 변수인 변수가 사용하는 메모리 영역 / 크기가 작고 관리(할당 및 반환)가 자동으로 이루어 짐
Heap		동적 할당할 수 있는 자유 메모리 영역 개발자 스스로 관리(수동) - 제일 큰 영역 할당 및 해제를 신경 써야 함
PE image (실행 영역)	Text section	C언어의 소스코드가 번역된 기계어가 저장된 메모리 영역 기본적으로 읽기전용 메모리
	Data section(Read only)	상수 형태를 기술하는 문자열(예: "Hello")이 저장된 메모리 영역
	Data section(Read/Write)	정적변수나 전역변수들이 사용하는 메모리 영역 별도로 초기화하지 않아도 0으로 초기화 관리는 자동이기 때문에 Heap 영역 메모리처럼 할당 및 해제를 신경 쓸 필요 없음

“ 변수는 컴파일 이후 메모리 주소 ”

메모리 지정 방식

- 운영 체제는 메모리 주소를 1Byte 단위로 관리
- 프로그래머는 메모리를 사용 하기 위해서는 사용 할 주소를 표기 해 주어야 한다.



2 진수	16 진수	2 진수	16 진수	2 진수	16 진수	2 진수	16 진수
0000	0	0100	4	1000	8	1100	C (12)
0001	1	0101	5	1001	9	1101	D (13)
0010	2	0110	6	1010	A (10)	1110	E (14)
0011	3	0111	7	1011	B (11)	1111	F (15)

- 직접 주소 방식과 간접 주소 방식 프로그래머는 사용 할 메모리를 지정 할 수 있음

2. 직접 주소 지정 방식

1. 포인터 1-1. 메모리 관리

“ 변수 사용은 C 언어에서 직접 주소 지정 방식 ”

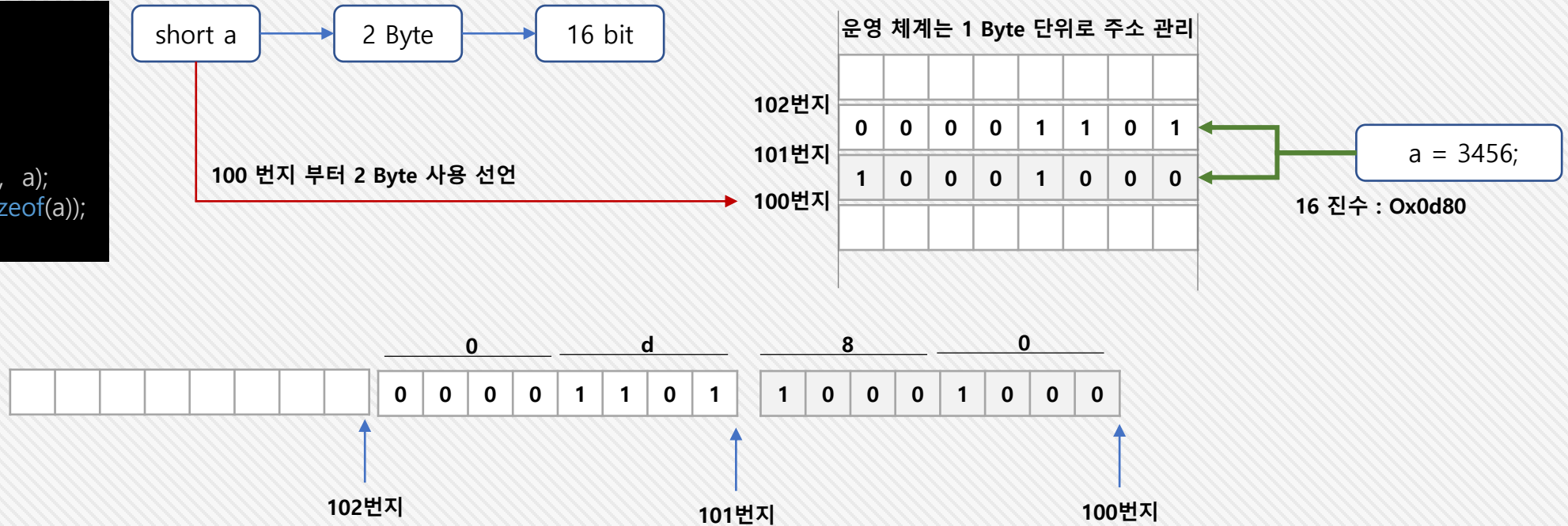
직접 주소 지정 방식

- 변수를 사용 하는 문법을 의미함.

```
#include <stdio.h>

int main() {
    short a = 3456;

    printf("a : %#x\n", a);
    printf("size : %d", sizeof(a));
}
```



2. 직접 주소 지정 방식

1. 포인터 1-1. 메모리 관리

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

```
void fun(b);
```

```
int main() {  
    int a = 3456;
```

1. 정수형 a 선언 및 초기화

```
    printf("a      : 0x%X\n", a);
```

```
    printf("a address : %p\n", &a);
```

```
    fun(a);
```

2. 함수 fun에 파라미터 전송

```
    printf("== after fun ==\n");
```

```
    printf("a      : 0x%X\n", a);
```

```
    printf("a address : %p\n", &a);
```

```
}
```

```
void fun(int b) {
```

```
    int c;
```

5. 정수형 변수 c 선언

```
    c = b;
```

6. b에 있는 값 c에 저장

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

```
    printf("b      : 0x%X\n", b);
```

```
    printf("b address : %p\n", &b);
```

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

```
    printf("c      : 0x%X\n", c);
```

```
    printf("c address : %p\n", &c);
```

```
    b += 1;
```

8. b에 있는 값 증감 계산

```
    printf("== b operator ==\n");
```

```
    printf("b      : 0x%X\n", b);
```

```
    printf("b address : %p\n", &b);
```

```
}
```

** main 함수 영역

0d

80

000000000061FE1C

3. 파라미터로 a 값 전달

** fun 함수 영역

0d

80

000000000061FDF0

7. B에 있는 값 c에 전달

000000000061FDDC

0d

80

000000000061FDDC

0d

81

000000000061FDF0

```
a      : 0xD80  
a address : 000000000061FE1C  
-----  
b      : 0xD80  
b address : 000000000061FDF0  
-----  
c      : 0xD80  
c address : 000000000061FDDC  
== b operator ==  
b      : 0xD81  
b address : 000000000061FDF0  
== after fun ==  
a      : 0xD80  
a address : 000000000061FE1C
```

3. 간접 주소 지정 방식

“ 참조 주소를 통해서 값의 관리 ”

간접 주소 지정 방식

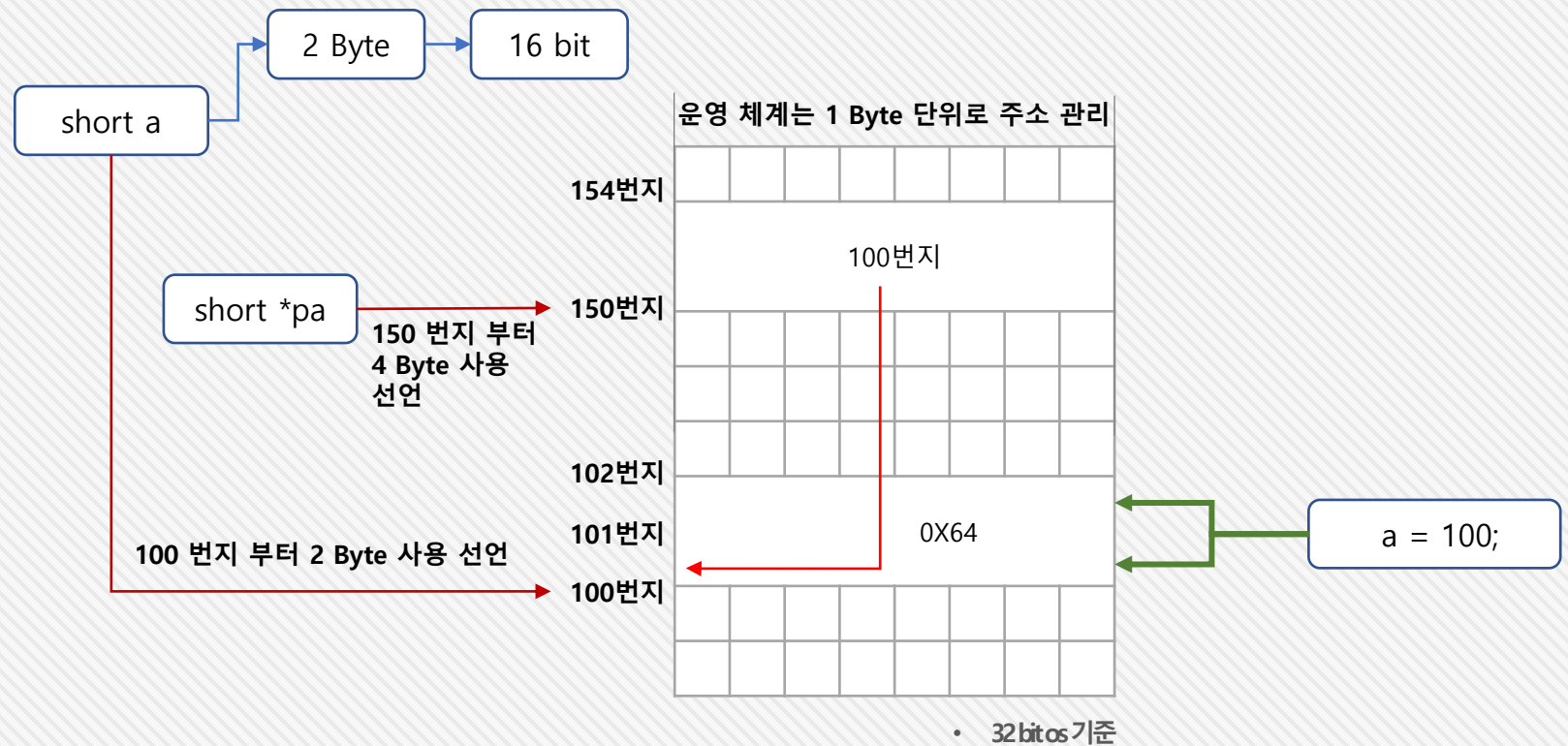
- 변수에 주소를 지정 하여 지정된 주소의 값을 관리.

```
#include <stdio.h>

void main() {
    short a;
    short *pa;

    pa = &a;
    a = 100;

    printf("a data    : 0x%X\n", a );
    printf("a address : %p\n", &a);
    printf("pa data    : 0x%X\n", pa );
    printf("pa address : %p\n", &pa);
    *pa += 1;
    printf("pa data    : 0x%X\n", pa );
    printf("pa address : %p\n", &pa);
    printf("a data    : 0x%X\n", a );
}
```



3. 간접 주소 지정 방식

1. 포인터 1-1. 메모리 관리

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

```
void fun(b);
```

```
int main() {  
    int a = 3456; 1. 정수형 a 선언 및 초기화
```

```
    printf("a      : 0x%X\n", a);  
    printf("a address : %p\n", &a);  
    fun(&a); 2. 함수 fun 호출  
    printf("== after fun ==\n");  
    printf("a      : 0x%x\n", a);  
    printf("a address : %p\n", &a);  
}
```

```
void fun(int *b) { 4. 파라미터 포인터 변수 b에 a 주소 저장
```

```
    int c; 5. 정수형 변수 c 선언  
    c = *b;  
    printf("-----\n");  
    printf("*b      : 0x%X\n", b);  
    printf("*b address : %p\n", &b);  
    printf("-----\n");  
    printf("c      : 0x%X\n", c);  
    printf("c address : %p\n", &c);  
    *b += 1;  
    printf("== b operator =\n");  
    printf("*b      : 0x%X\n", b);  
    printf("*b address : %p\n", &b);  
}
```

** main 함수 영역

000000000061FE1C

0xD80

** fun 함수 영역

000000000061FDF0

0x61FE1C

000000000061FDDC

0xD80

9. 포인터 p에 저장 되어 있는 주소에
있는 값을 읽어서 +1 후 포인터 p에
저장된 주소를 찾아가서 값을 변경 함

3. 파라미터로 a 주소 전달

7. 주소로 이동

8. 값 저장

```
a      : 0xD80  
a address : 000000000061FE1C  
-----  
*b      : 0x61FE1C  
*b address : 000000000061FDF0  
-----  
c      : 0xD80  
c address : 000000000061FDDC  
== b operator =  
*b      : 0x61FE1C  
*b address : 000000000061FDF0  
== after fun ==  
a      : 0xd81  
a address : 000000000061FE1C
```

“ Pointer는 참조 주소 방식으로 메모리 주소에 연결된 데이터를 관리 함 ”

Pointer

- 메모리 주소만 저장 하는 변수
- “*”를 이용해서 변수를 선언 함
- 4 byte (32 bit OS), 8 byte (64 bit OS) 의 크기 [일반 변수는 데이터의 크기]
- 자료형 : Pointer 변수 저장된 주소의 저장 될 값

자료형

*

변수명 ;

➤ 선언한 변수의 메모리 주소 얻기

```
short var;           // 변수 선언
short *ptrVar;       // 포인터 변수 선언

ptrVar = &var;       // 변수의 메모리 주소를 포인터 변수에 저장

var = 0x64;

printf("var   Data   : 0x%X\n", var);    // var 변수에 저장된 값
printf("var   Address : %p\n", ptrVar);  // var 변수 메모리 주소
printf("ptrVar Address : %p\n", &ptrVar); // ptrVar 변수 메모리 주소
printf("ptrVar Data   : 0x%X\n", ptrVar); // ptrVar 변수에 저장된 값

*ptrVar += 1; // 포인터 변수에 저장된 주소에 해당 되는 값을 변경
printf("var   Data   : 0x%X\n", var);
```

```
var      Data      : 0x64
var      Address   : 000000000061FE1E
ptrVar   Address   : 000000000061FE10
ptrVar   Data      : 0x61FE1E
var      Data      : 0x65
```

short *ptr;

112번지

0x1F10

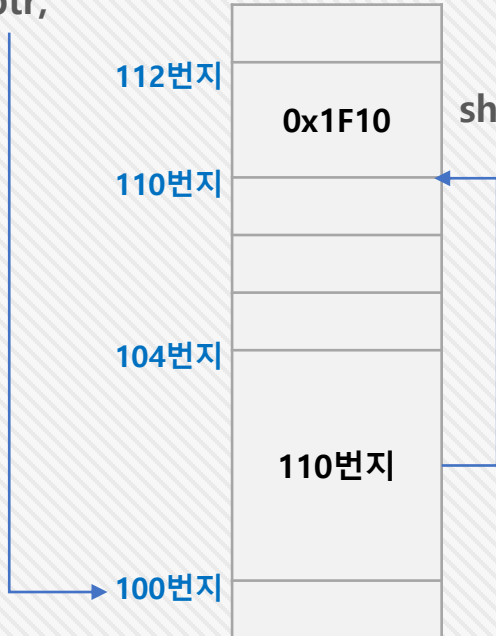
short num = 0x1F10;

110번지

104번지

110번지

100번지



1. 포인터

1. 포인터 1-2. 포인터

- 포인터 변수에 변수를 할당 하면 컴파일 오류 발생: **자료형 불일치**

```
short var  
short *ptrVar;  
ptrvar = var;
```

error: 'ptrvar' undeclared (first use in this function); did you mean 'ptrVar'?

- 변수, 주소연산자, 역참조연산자, 포인터의 차이

❖ **int num = 0;**

&num

주소연산자 : num 변수의 메모리 주소

num

변수 : num 변수 값을 get/set

❖ **int *ptrNum;**

*ptrNum = 1

역참조연산자 : 포인터 변수 (ptrNum)에 저장된 주소에 접근 하여 값을 get/set

ptrNum

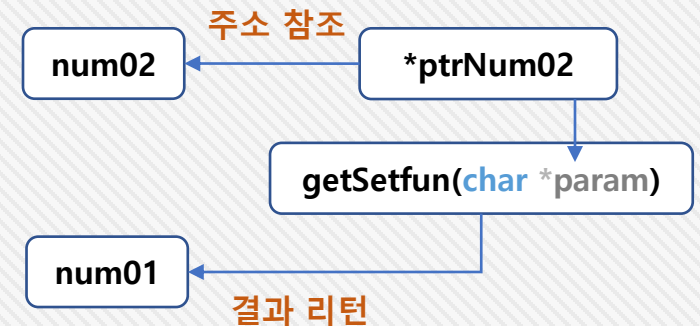
포인터 : 포인터 변수(ptrNum)의 주소

- 직접 주소 방식의 한계를 간접 주소 (pointer)을 사용 하여 극복 함

```
#include <stdio.h>  
int getSetfun (int *param);
```

```
void main() {  
    int num01, num02, *ptrNum02;  
    ptrNum02 = &num02;  
    num02 = 0;  
    num01 = getSetfun(ptrNum02);  
    printf("num01 : %d\n", num01);  
    printf("num02 : %d", num02);  
}
```

```
int getSetfun(int *param){  
    int num = 1;  
    *number += 2;  
    return num;  
}
```



1. 포인터

1. 포인터 1-2. 포인터

➤ 변수 값 교환

- 직접 참조로 하면 결과 반환시 point 사용 해야함

```
#include <stdio.h>

void funcDirectSwapMain();
int* funcDirectSwap(int a, int b);
void funcIndirectSwapMain();
void funcIndirectSwap(int *a, int *b);

void main() {
    funcDirectSwapMain();
    funcIndirectSwapMain();
}
```

```
void funcDirectSwapMain() {
    int a = 2;
    int b = 10;
    printf("DirectSwap=====\\n");
    printf("before a = %d , b = %d\\n", a, b);
    int *numChg = funcDirectSwap( a, b);
    a = numChg[0];
    b = numChg[1];
    printf("after a = %d , b = %d\\n", a, b);
}
```

```
int* funcDirectSwap(int a, int b) {
    static int result[2] ;
    int temp = b;
    b = a;
    a = temp;
    result[0] = a;
    result[1] = b;
    return result;
}
```

```
void funcIndirectSwapMain() {
    int a = 2;
    int b = 10;
    printf("IndirectSwap=====\\n");
    printf("before a = %d , b = %d\\n", a, b);
    funcIndirectSwap(&a, &b);
    printf("after a = %d , b = %d\\n", a, b);
}
```

```
void funcIndirectSwap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```

```
d:\works\c_basic\src\pointer>.\"24_changeVariable.exe"
DirectSwap=====
before a = 2 , b = 10
after a = 10 , b = 2
IndirectSwap=====
before a = 2 , b = 10
after a = 10 , b = 2
```


1. 포인터

1. 포인터 1-2. 포인터

➤ const 키워드를 Pointer 사용

`int *const ptr;`

ptr가 가지고 있는 주소를 변경 금지

```
void funcNoChangeConstPointer (){
    int num01 = 10, num02 = 5;
    int *const ptr = &num01;
    ptr = &num02;
}
```

```
24_constPointer.c: In function 'funcNoChangeConstPointer':
24_constPointer.c:15:9: error: assignment of read-only variable 'ptr'
    ptr = &num02;
    ^
```

`const int *ptr;`

ptr가 가지고 있는 주소의 값을 변경 변경 금지

```
void funcNoChangePointerValue(){
    int num01 = 10;
    const int *ptr = &num01;
    *ptr = 30;
}
```

```
24_constPointer.c: In function 'funcNoChangePointerValue':
24_constPointer.c:21:10: error: assignment of read-only location '*ptr'
    *ptr = 30;
    ^
```

`const int * const ptr;`

ptr가 가지고 있는 주소, 주소의 값 변경 금지

```
void funcNoChangeAll() {
    int num01 = 10, num02 = 5;
    const int *const ptr = &num01;
    *ptr = 30;
    ptr = &num02;
}
```

```
24_constPointer.c:27:10: error: assignment of read-only location '*ptr'
    *ptr = 30;
    ^
24_constPointer.c:28:9: error: assignment of read-only variable 'ptr'
    ptr = &num02;
    ^
```

1. 포인터

1. 포인터 1-2. 포인터

➤ void 포인터

void *포인터 이름

- 대상의 크기가 정해져 있지 않는 pointer
- 모든 자료형을 설정 할 수 있음
- 시작 주소만 있고 끝 주소를 모를 때 사용
- 사용할 크기를 반드시 표기 해야 한다 => 형 변환 문법 사용

```
void main() {  
    int num = 10;  
    void *voidPtrNum = &num;  
    *voidPtrNum = 20;  
}
```

```
26_VoidPointer.c:6:5: warning: dereferencing 'void *' pointer  
    *voidPtrNum = 20;  
    ^~~~~~  
26_VoidPointer.c:6:17: error: invalid use of void expression  
    *voidPtrNum = 20;  
    ^
```

```
void main() {  
    int num = 10;  
    void *voidPtrNum = &num;  
    *(int *)voidPtrNum = 20;  
}
```

- 대상 메모리의 크기를 조절

```
#include <stdio.h>  
  
int funGetData(void *ptr, char type);  
  
void main() {  
    int num = 1234567890;  
    printf("%d\n", funGetData(&num, 'c'));  
    printf("%d\n", funGetData(&num, 's'));  
    printf("%d\n", funGetData(&num, 'i'));  
}
```

```
int funGetData(void *ptr, char type) {  
    int data = 0;  
  
    if (type == 'c') data = *(char *) ptr;  
    else if (type == 's') data = *(short *) ptr;  
    else data = *(int *) ptr;  
  
    return data;  
}
```

```
d:\works\c_basic\src\pointer>.\"26_VoidPointer.exe"  
-46  
722  
1234567890
```

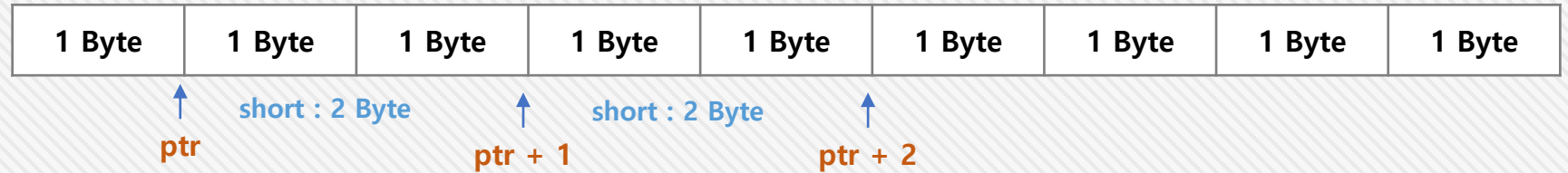
1. 포인터

1. 포인터 1-2. 포인터

➤ 포인터 주소 연산의미

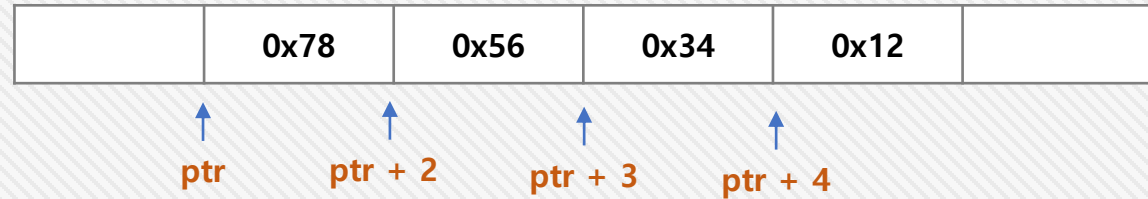
- 선언한 자료형 만큼 참조 주소 값 변경됨

```
short data = 10;  
short *ptr = &data;  
ptr = ptr + 1;
```



➤ 포인터 대상의 크기가 실제 크기와 틀리면 포인터 선언 시 선언한 자료형 만큼 자료를 가지고 온다.

```
#include <stdio.h>  
void main() {  
    int i;  
    int num = 0x12345678;  
    char *ptr = (char *)&num;  
    for ( i = 0; i < 4; i++) {  
        printf("0x%X ", *ptr);  
        ptr++;  
    }  
}
```



```
d:\works\c_basic\src\pointer>.\"27_pointer0p.exe"  
0x78 0x56 0x34 0x12
```

Content

III. C언어 고급 기본

1. 메모리 할당
2. 구조체
3. 파일 입출력
4. 다차원 포인터
5. 함수 포인터