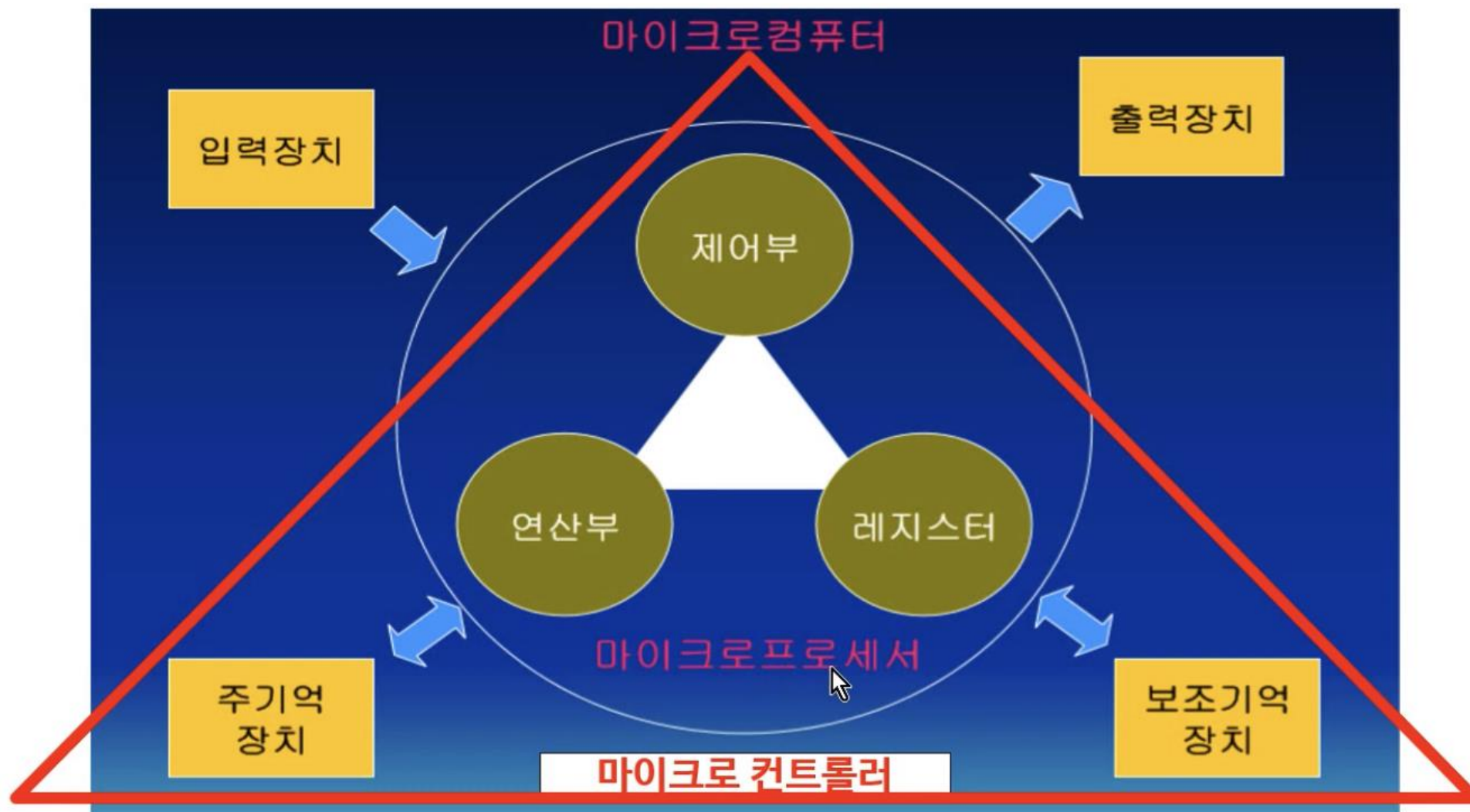




마이크로 프로세서

마이크로프로세서 vs 마이크로컨트롤러 vs 마이크로컴퓨터



마이크로프로세서 vs 마이크로컨트롤러

- ▶ 마이크로프로세서(MicroProcess) : 컴퓨터 중앙 처리 장치(CPU)의 핵심 기능을 통합한 집적 회로(IC)
- ▶ 마이크로컨트롤러 (MicroController Unit, MCU) : 마이크로프로세서에 연결되는 RAM(SRAM), ROM(Flash)과 함께 Timer, SPI, I2C, ADC, UART 등이 하나의 칩 안에 포함된 것
- ❖ 마이크로프로세서와 마이크로컨트롤러를 최근 들어 구분없이 사용되나 마이크로프로세서는 단독으로 동작하지 못하는 CPU 외부에 메모리와 주변장치들이 연결되어 동작하는 것이고, 마이크로컨트롤러는 CPU 내부에 메모리와 주변장치들이 포함되어 있어 단독 동작이 가능한 것

마이크로프로세서의 특징

- ▶ 저비용 : 집적 회로 기술로 저렴한 비용으로 이용가능하므로 컴퓨터 시스템의 비용 감소 가능
- ▶ 고속 : 초당 수백만개의 명령을 실행할 정도 빠른 속도로 작동
- ▶ 작은 사이즈 : 전체 컴퓨터 시스템의 크기 감소 가능
- ▶ 다재다능 : 용도가 매우 다양하며, 동일한 칩은 프로그램을 간단히 변경하여 다른 용도로 사용 가능
- ▶ 저전력 소비 : 다른 시스템에 비해 전력 소비
- ▶ 낮은 발열량 : 진공 튜브 장치에 비해 작은 열 방출
- ▶ 높은 신뢰성 : 반도체 기술에 의해 낮은 고장율을 가짐
- ▶ 휴대성 : 작은 크기와 낮은 전력 소비 때문에 휴대 가능

마이크로프로세서의 종류

▶ 8051

- x86 CPU 생산업체인 인텔(Intel)에서 만든 MCU
- TI 사의 TMS1000이라는 MCU와 더불어 1975년에 개발된 초창기 MCU

▶ PIC(Peripheral Interface Controller)

- MicroChip Technology사에서 만든 MCU
- 산업용으로 많이 사용되어 주변의 가전제품 속에서 쉽게 볼 수 있음

▶ AVR(Advanced Virtual RISC)

- Atmel사에서 만든 MCU
- 개발 환경이 잘 구성되어 있으며 새로운 프로그램을 기록하는 방법이 용이
- AVR은 PIC보다 처리 속도가 빠르며, 8051보다 학습자료가 풍부하기 때문

▶ ARM(Advanced RISC Machine)

- 저전력의 MCU로서 거의 PC에 버금가는 활용도를 자랑
- OS 탑재가 가능하여 많은 제조사에서 ARM 회로도를 사용하여 MCU 제작
- 기존의 다른 MCU와 다른점인데 ARM은 회로설계만 하고 제조를 하지 않음

마이크로프로세서의 종류

▶ RISC / CISC 개념

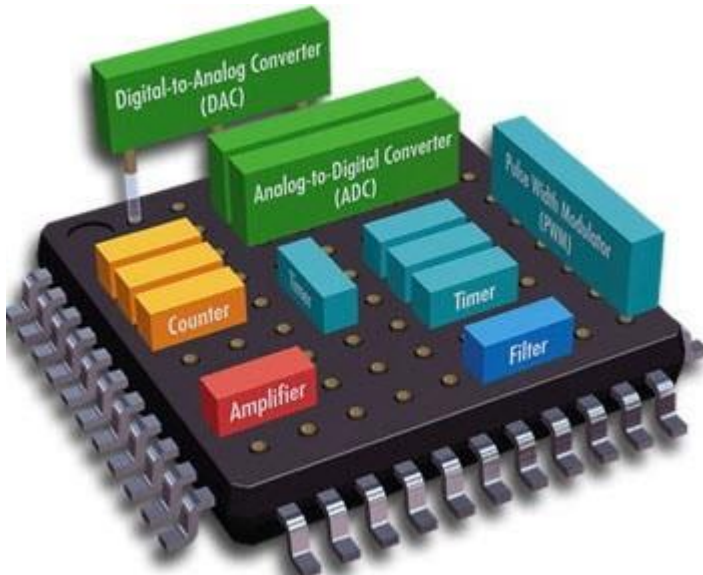
- CPU(중앙처리장치) 를 설계하는 방식
- CPU가 작동하려면 프로그램이 있어야 하고 명령어를 주입해서 설계함.
- RISC : 명령어가 H/W 적인 방식
- CISC : 명령어가 S/W 적인 방식

▶ RISC / CISC 비교

	CISC (Complex Instruction Set Computer)	RISC(Reduced Instruction Set Computer)
명령어의 수	많다	적다
레지스터	적다	많다
처리속도	느리다	빠르다
설계(내부구조)	복잡하다	간단하다
전력소모	많다	적다

임베디드 시스템

- ▶ 임베디드 시스템(Embedded System, 내장형 시스템)은 기계나 기타 제어
가 필요한 시스템에 대해, 제어를 위한 특정 기능을 수행하는 컴퓨터 시
스템으로 장치 내에 존재하는 전자 시스템
- ▶ 임베디드 시스템은 전체 장치의 일부분으로 구성되며 제어가 필요한 시
스템을 위한 두뇌 역할을 하는 특정 목적의 컴퓨터 시스템



WSL & Ubuntu 20.04 설치

- ▶ 명령 프롬프트(cmd)를 관리자 권한(administrator)으로 실행
 - > wsl --list --online // 사용 가능한 배포판 목록 확인
 - > wsl --install -d Ubuntu-20.04 // 배포판 설치
- ▶ Ubuntu 20.04 install을 위해서는 리부팅 필요(1번 또는 2번)
- ▶ 리부팅 후 새로운 윈도우가 열리고 cmd 화면(Ubuntu-20.04) 자동 실행
- ▶ user account 생성
 - > id: <user_id>
 - > passwd: <user_passwd>
 - > repasswd: <user_passwd>

Ubuntu 20.04 Settings

- ▶ system update
 - > sudo apt update
 - > sudo apt upgrade
- ▶ net-tools install
 - > sudo apt install net-tools // ifconfig 명령을 사용하기 위해 설치
- ▶ gcc & gdb install
 - > sudo apt install gcc gdb // C 컴파일러, 디버거 설치

리눅스의 특징

- ▶ 이식성과 확장성이 용이
- ▶ 텍스트 모드 중심의 관리와 다양한 관리 환경 제공
- ▶ 풍부한 소프트웨어 개발 환경 제공
- ▶ 다양한 네트워크 서비스 및 작업환경 지원
- ▶ 뛰어난 안정성
- ▶ 시스템 보안성
- ▶ 폭넓은 하드웨어 장치 지원
- ▶ 시스템의 높은 신뢰성
- ▶ 가격 대비 탁월한 성능(무료)

리눅스 명령(1)

- ▶ ls : 리스트출력
- ▶ ls -l : 리스트 출력(사용권한, 소유자, 그룹, 크기, 날짜 등 상세정보 출력)
- ▶ ls -a : 보이지 않는 리스트까지 출력
- ▶ cd [폴더명] : 입력한 폴더로 이동
- ▶ cd ~ : Desktop 폴더로 이동
- ▶ cd .. : 한단계 상위폴더로 이동
- ▶ mkdir [폴더명] : 폴더 생성(띄어쓰기 후 여러개의 폴더 생성 가능)
- ▶ touch [파일명] : 입력한 파일생성(띄어쓰기 후 여러개의 파일 생성 가능)
- ▶ mv [파일명] [폴더명/파일명] : 파일을 입력한 폴더로 이동
- ▶ mv [파일명] [변경될파일명] : 현재파일을 입력한 파일명으로 변경

리눅스 명령(2)

- ▶ `pwd` : 현재 작업중인 폴더의 절대경로가 출력
- ▶ `cat [파일명]` : 파일내용확인
- ▶ `cp [폴더명/파일명] [복제될파일명]` : 폴더의 파일을 현재 폴더에 복제
- ▶ `rm [파일명]` : 파일 삭제(폴더불가능)(띄어쓰기 후 여러개 삭제 가능)
(*.[파일종류]를 입력하면 해당 파일 종류 모두 삭제)
- ▶ `rmdir [폴더명]` : 폴더삭제(단, 빈폴더가 아닐 경우 삭제 안됨)
- ▶ `rm -rf [폴더명]` : 비어있지 않은 폴더 삭제
- ▶ `[명령어] --help` : 명령어 도우미

리눅스 명령(3)

- ▶ `$ sudo apt` // 프로그램 설치, 검색, 제거 등
- ▶ `$ sudo apt update` // 설치 가능한 패키지 리스트를 최신화
- ▶ `$ sudo apt upgrade` // list 안에 있는 패키지에 대한 최신 버전 재설치
- ▶ `$ sudo apt install` // 사용자가 원하는 패키지 설치

VSCode 설치

- ▶ <https://code.visualstudio.com/> 접속

1. Install mode

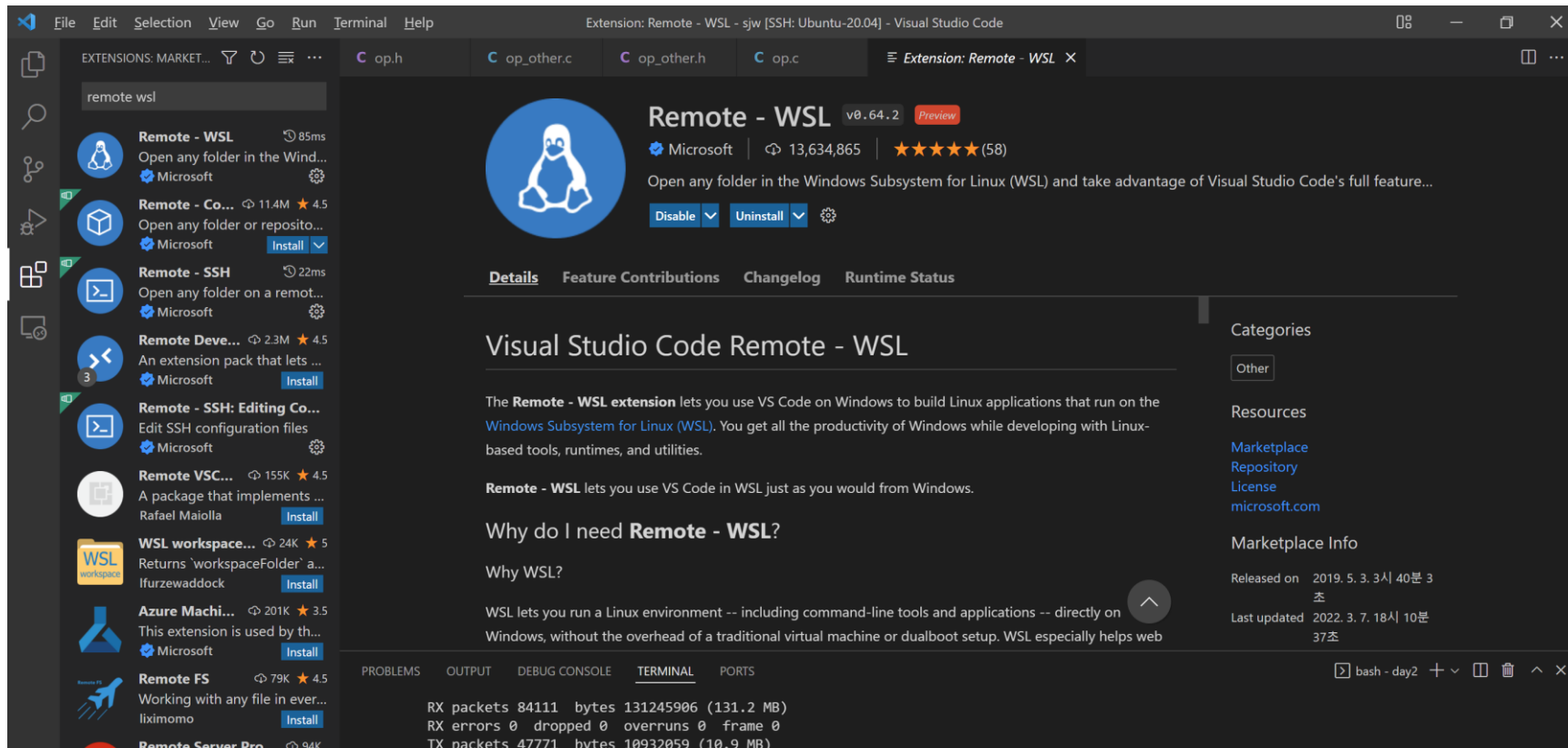
- ▶ 다운로드 : system Installer 64bit
- ▶ 다운로드 후 설치

2. portable mode

- ▶ 다운로드 : .zip 64bit
- ▶ 다운로드 파일 압축 풀기
- ▶ 압축을 푼 폴더의 이름을 VSCode로 변경
- ▶ VSCode 하위에 data 폴더 생성
- ▶ VSCode 폴더를 C:\로 이동

VSCode에서 Remote-WSL 설치

▶ extension tab에서 Remote-WSL 추가 설치

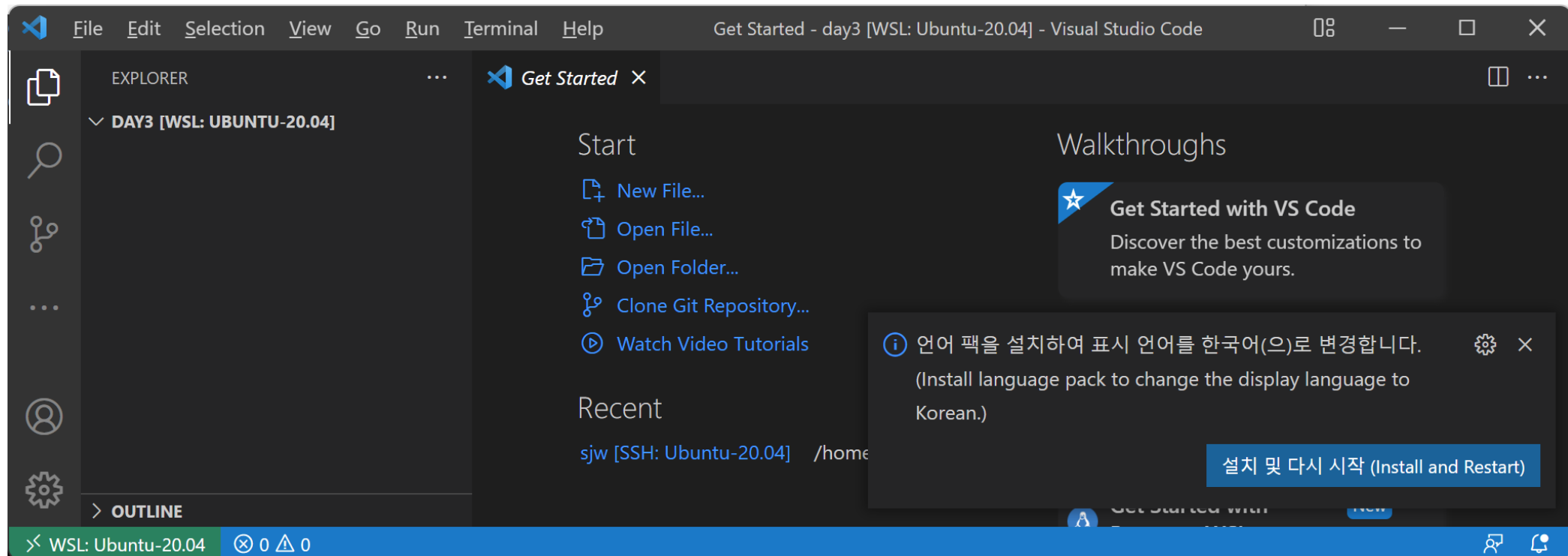


Ubuntu에서 VSCode 실행

▶ Ubuntu에서 VSCode 실행

\$ code . // VSCode 실행

▶ Ubuntu상에서 vscode 실행한 상태(좌측 하단에 WSL:Ubuntu-20.04 표시)



Portable Mode에서 VSCode 자동실행(1)

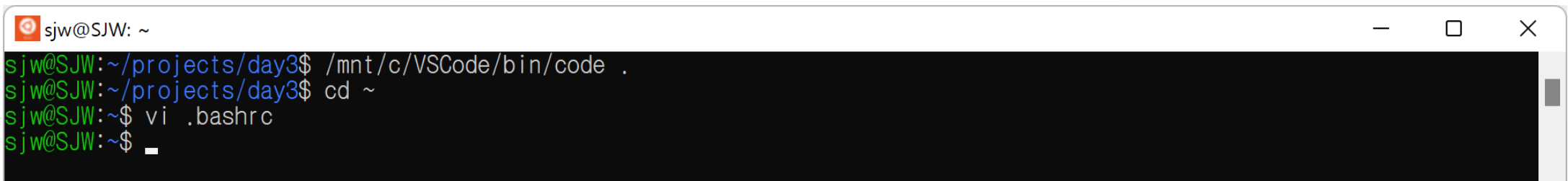
- ▶ Ubuntu에서 아래 경로를 입력하면 VSCode 실행

\$ /mnt/c/VSCode/bin/code .

※ 참고) 윈도우 경로 : c/VSCode/bin/code

- ▶ Ubuntu에서 VSCode가 자동 실행되도록 환경 변수 등록
(반드시 home 디렉토리에서 실행 : cd ~)

\$ vi .bashrc // vi편집기를 이용해서 PATH 수정

A terminal window titled 'sjw@SJW: ~' with standard window controls. It shows a sequence of commands and their outputs: 1. 'sjw@SJW:~/projects/day3\$ /mnt/c/VSCode/bin/code .' which results in a new prompt 'sjw@SJW:~/projects/day3\$'. 2. 'sjw@SJW:~/projects/day3\$ cd ~' which results in a new prompt 'sjw@SJW:~\$'. 3. 'sjw@SJW:~\$ vi .bashrc' which results in a new prompt 'sjw@SJW:~\$'. 4. A final prompt 'sjw@SJW:~\$' with a cursor, indicating the user has exited the editor.

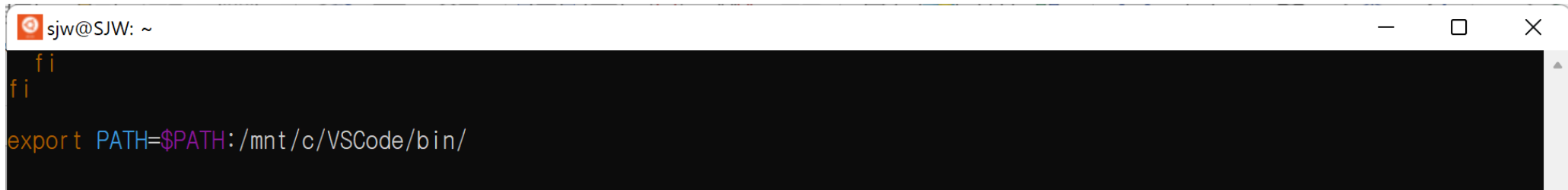
```
sjw@SJW: ~
sjw@SJW:~/projects/day3$ /mnt/c/VSCode/bin/code .
sjw@SJW:~/projects/day3$ cd ~
sjw@SJW:~$ vi .bashrc
sjw@SJW:~$
```

portable mode에서 VSCode 자동실행(2)

- ▶ 리눅스 환경변수에 등록하기 위해 bashrc에 PATH 기입 후 저장
(참고) 전체 환경 변수 확인 : `$ echo $PATH`

- ▶ .bashrc에서 맨 아래 다음의 PATH 설정 추가

```
export PATH=$PATH:/mnt/c/VSCode/bin/
```

A terminal window with a black background and white text. The title bar shows 'sjw@SJW: ~'. The terminal content shows the command 'export PATH=\$PATH:/mnt/c/VSCode/bin/' being entered. There are some faint, partially visible characters 'fi' above the command.

```
sjw@SJW: ~  
fi  
fi  
export PATH=$PATH:/mnt/c/VSCode/bin/
```

※ 다른 셸(예 .zshrc)을 사용할 경우에는 반드시 추가 PATH 설정할 것

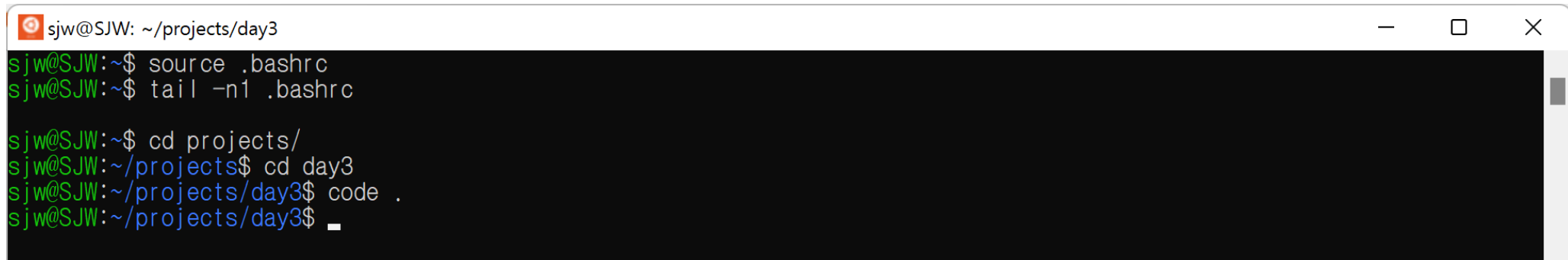
portable mode에서 VSCode 자동실행(3)

▶ Ubuntu에서 다음의 명령 입력

\$ source .bashrc // 실행된 환경에서 다시 실행(리부팅하지 않아도 됨)

\$ tail -n1 ~/.bashrc // 셸의 마지막에 수정된 내용 확인

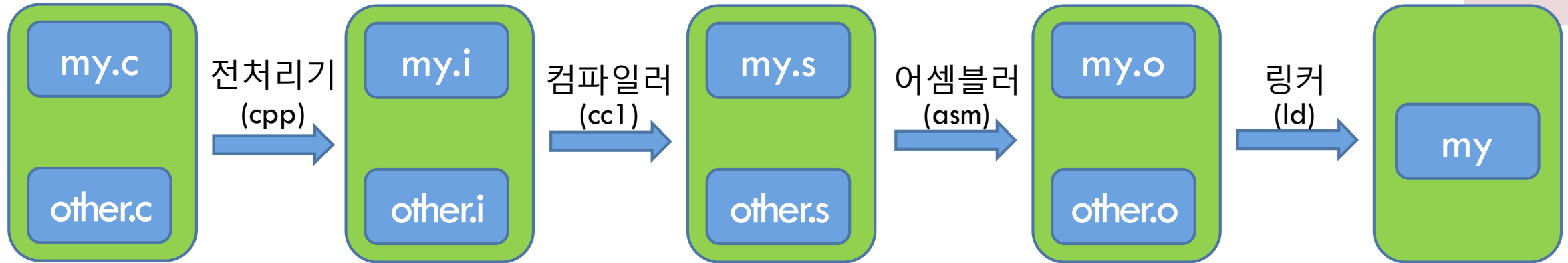
\$ code . // VSCode 실행



```
sjw@SJW: ~/projects/day3
sjw@SJW:~$ source .bashrc
sjw@SJW:~$ tail -n1 .bashrc

sjw@SJW:~$ cd projects/
sjw@SJW:~/projects$ cd day3
sjw@SJW:~/projects/day3$ code .
sjw@SJW:~/projects/day3$ _
```

C 프로그램 생성 과정



- ▶ 전처리기 : 소스 파일(*.c)에 gcc를 동작시키면 가장 먼저 전처리기 cpp가 동작하고, cpp는 소스 파일(*.c)의 #include와 #define과 같은 전처리기 부분을 처리(즉, 필요한 헤더 파일을 삽입하고 실행문장의 매크로를 상수로 변환)
- ▶ 컴파일러 : 전처리된 파일(*.i)로부터 어셈블리어로 된 파일(*.s)을 생성
- ▶ 어셈블러 : 어셈블리어로 된 파일(*.s)을 기계가 직접 이해할 수 있는 기계어로 된 오브젝트 파일(*.o)로 변환
- ▶ 링커 : 오브젝트 파일(*.o)은 printf, scanf와 같은 라이브러리 함수가 없기에 실행될 수 없고, 여러 파일로 이루어진 프로그램의 경우 파일 간 연결이 이루어지지 않아 실행되지 않는데, 이 때 링커를 활용해 라이브러리 함수와 오브젝트파일들을 연결해 실행 파일을 생성

gcc 실행방법

- ▶ 컴파일 방법

\$ gcc 소스파일이름

(a.out 파일 생성, a.out을 실행하기 위해선 \$./a.out)

- ▶ 컴파일 할때 출력 파일 이름 지정 방법

\$ gcc -o 출력파일이름 소스파일이름

(ex. \$ gcc -o file file.c)

- ▶ 여러 파일을 동시에 컴파일 하는 방법

\$ gcc -o 출력파일이름 소스파일이름1, 소스파일이름2

(ex. \$ gcc -o file file1.c file2.c)

포인터(1) – 포인터 변수 선언

▶ pointer1.c

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

```
int main(){  
    int *ptr;           // 포인터 변수 선언  
    int num = 10;       // int형 변수를 선언하고 10 저장  
    ptr = &num;         // ptr에 num의 메모리 주소 저장  
    printf("%p\n", ptr); // ptr에 저장된 num의 메모리 주소 출력  
    printf("%p\n", &num); // num의 메모리 주소 출력  
    return 0;  
}
```

참고) 컴파일 : gcc -o pointer1 pointer1.c

포인터(2) – 역참조1

▶ pointer2.c

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

```
int main(){  
    int *ptr;           // 포인터 변수 선언  
    int num = 10;       // int형 변수를 선언하고 10 저장  
    ptr = &num;         // ptr에 num의 메모리 주소 저장  
    printf("%d\n", *ptr); // 역참조 연산자로 num의 메모리 주소에 접근하여 값을 가져옴  
    return 0;  
}
```

참고) 컴파일 : gcc -o pointer2 pointer2.c

포인터(3) – 역참조2

▶ Pointer2-1.c

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

```
int main(){  
    int *ptr;           // 포인터 변수 선언  
    int num = 10;       // int형 변수를 선언하고 10 저장  
    ptr = &num;         // ptr에 num의 메모리 주소 저장  
    *ptr = 20;          // 역참조 연산자로 메모리 주소에 접근하여 20을 저장  
    printf("%d\n", *ptr); // 역참조 연산자로 메모리 주소에 접근하여 값 20을 출력  
    printf("%d\n", num);  // 실제 num의 값도 20으로 바뀌어 출력  
    return 0;  
}
```

참고) 컴파일 : gcc -o pointer2-1 pointer2-1.c

포인터(4) – 이중포인터

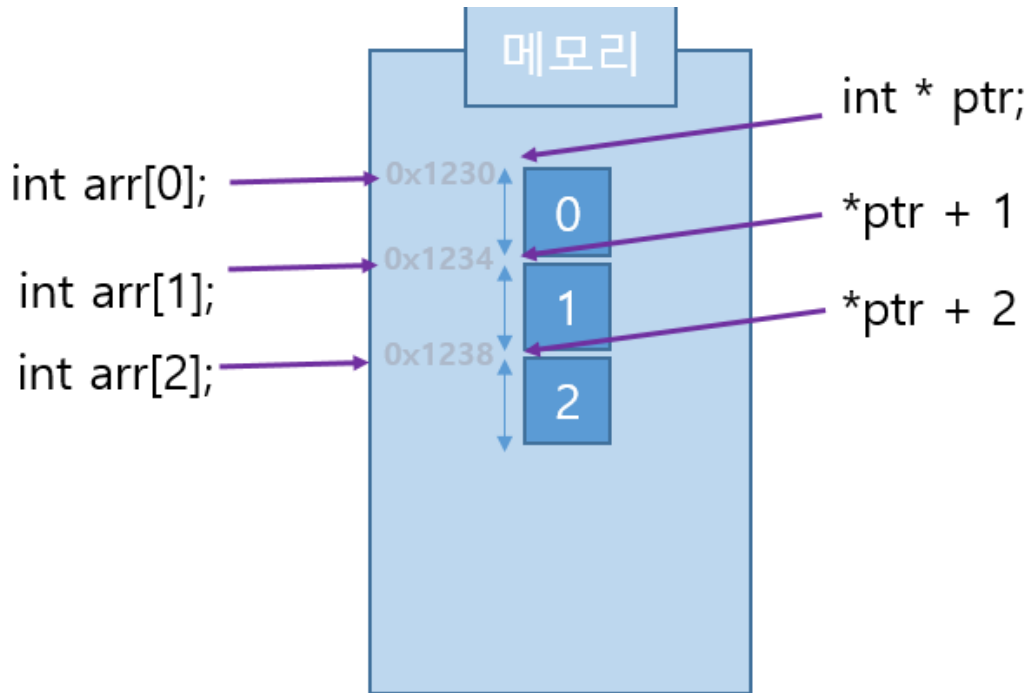
▶ pointer3.c

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

```
int main() {  
    int *ptr1;           // 단일 포인터 선언  
    int **ptr2;          // 이중 포인터 선언  
    int num = 10;  
    ptr1 = &num;         // num의 메모리 주소를 ptr1에 저장  
    ptr2 = &ptr1;        // ptr1의 메모리 주소를 ptr2에 저장  
    printf("%d\n", **ptr2); // 포인터를 두 번 역참조하여 num의 메모리 주소에 접근하여 10출력  
    return 0;  
}
```

참고) 컴파일 : gcc -o pointer3 pointer3.c

배열과 포인터(1)



주소가 순차적으로 배열되어있음.

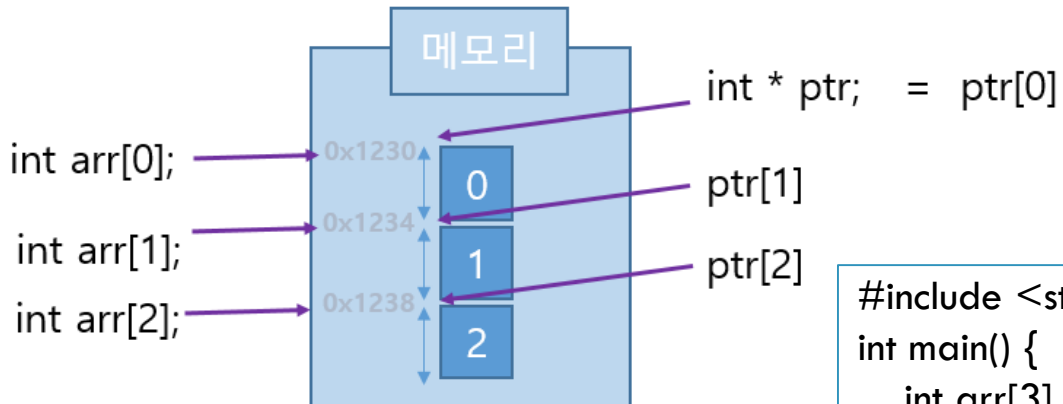
Ex) int형인 경우 4byte 이므로 주소는 4byte값 차이로 나란히 배열

- ▶ `int * ptr = &arr[0];` 이라고 할때 `arr`의 첫번째 주소를 `ptr`으로 가리키는 것
- ▶ `arr` 자체가 주소값이므로 `int * ptr = arr` 으로도 사용 가능
- ▶ `*ptr` 은 배열의 첫번째 항의 값을 출력함으로 0을 리턴
- ▶ `*(ptr + 1)` 은 `ptr`의 주소값에서 +1 (int형이므로 4byte) 뒤로 더한곳으로 주소를 가리킴

배열과 포인터(2)

```
#include <stdio.h>
int main() {
    int arr[3] = { 0, 1, 2 };           // 배열 초기값 출력 해보기
    printf("arr[0] : %d\n", arr[0]);
    printf("arr[1] : %d\n", arr[1]);
    printf("arr[2] : %d\n", arr[2]);
    int* ptr = &arr[0];                // 배열을 조작할 포인터 선언, arr[0] 첫번째 주소를 가리킴.
    *ptr = 10;                          // 첫번째 값 변경
    *(ptr + 1) = 30;                    // 두번째 값 변경, 자료형이 int이므로 1씩 증가하면 4byte씩 이동
    *(ptr + 2) = 300;                   // 세번째 값 변경
    printf("변경 후 배열 출력\n");      // 배열 출력
    printf("arr[0] : %d\n", arr[0]);
    printf("arr[1] : %d\n", arr[1]);
    printf("arr[2] : %d\n", arr[2]);
    return 0;
}
```

배열과 포인터(3)



- ▶ 선언한 포인터가 배열의 첫번째 칸을 가리키면, 포인터도 마치 배열처럼 사용 가능

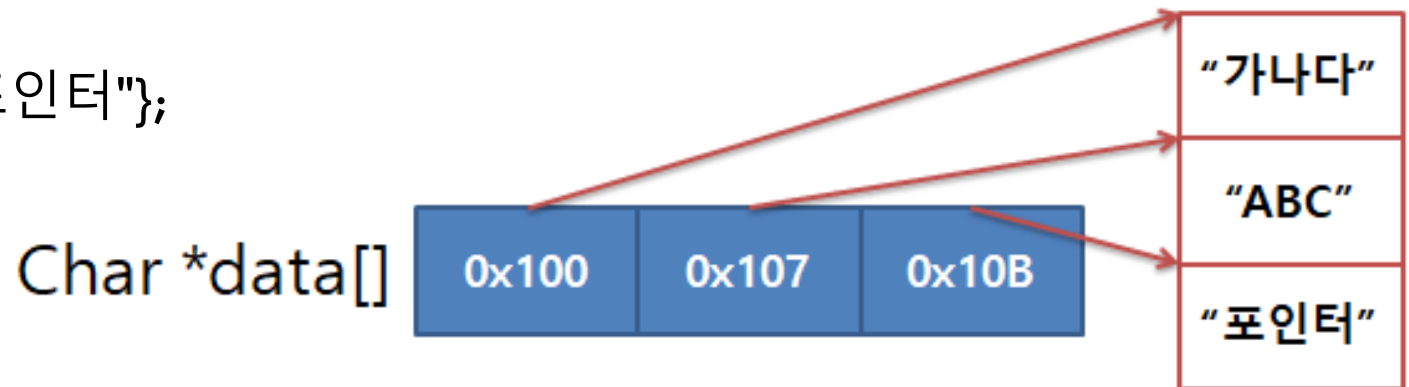
```
#include <stdio.h>
int main() {
    int arr[3] = { 0, 1, 2 };
    printf("arr[0] : %d\n", arr[0]); // 배열 초기값 출력
    printf("arr[1] : %d\n", arr[1]);
    printf("arr[2] : %d\n", arr[2]);
    int* ptr = &arr[0]; // 배열을 조작할 포인터 선언, arr[0] 첫번째 주소를 가리킴.
    ptr[0] = 50;        // 첫번째 값 변경
    ptr[1] = 70;        // 두번째 값 변경, 자료형이 int이므로 1씩 증가하면 4byte씩 이동
    ptr[2] = 100;       // 세번째 값 변경
    printf("변경 후 배열 출력\n"); // 배열 출력해
    printf("arr[0] : %d\n", arr[0]);
    printf("arr[1] : %d\n", arr[1]);
    printf("arr[2] : %d\n", arr[2]);
    return 0;
}
```

포인터 배열(1)

- ▶ 배열의 요소가 포인터들로 이루어짐
ex) 배열 요소의 자료형이 `char*` (포인터)이고, 요소 개수가 3개일 때 : `char* data[3];`
- ▶ 포인터 배열은 문자열 등과 같이 큰 길이에 데이터들을 포인터로 가리키게 하고, 해당 포인터들을 다시 배열로 묶어서 관리하기 쉽게 할 때 사용
- ▶ 문자열 여러개를 각각 포인터가 가리키게 하고 그 포인터들을 하나의 배열로 묶어서 관리해서 사용하기에도 쉽고 가독성도 좋음
- ▶ 만약에 따로따로 하나의 `char*` 로 할당을 했다면 `for`문을 사용하기도 힘들었을 것이다.

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

```
int main() {  
    char *data[] = {"가나다", "ABC", "포인터"};  
    for (int i = 0; i < 3; i++) {  
        printf("%s\n", data[i]);  
    }  
    return 0;  
}
```



포인터 배열(2)

```
#include<stdio.h>
int main(void) {
    char* arr[3]; //포인터 배열 선언.
    int i;
    arr[0] = "BlockDMask"; //arr[0]은 -> 문자열 주소를 가리킴
    arr[1] = "C Programming"; //arr[1]은 -> 문자열 주소를 가리킴
    arr[2] = "point_arr"; //arr[2]은 -> 문자열 주소를 가리킴
    for (i = 0; i < 3; i++) {
        printf("arr[%d] -> %s\n", i, arr[i]);
    }
    return 0;
}
```