Busan Software Meister High School

MICROPROCESSOR

2309양유빈

20230309

마이크로프로세서

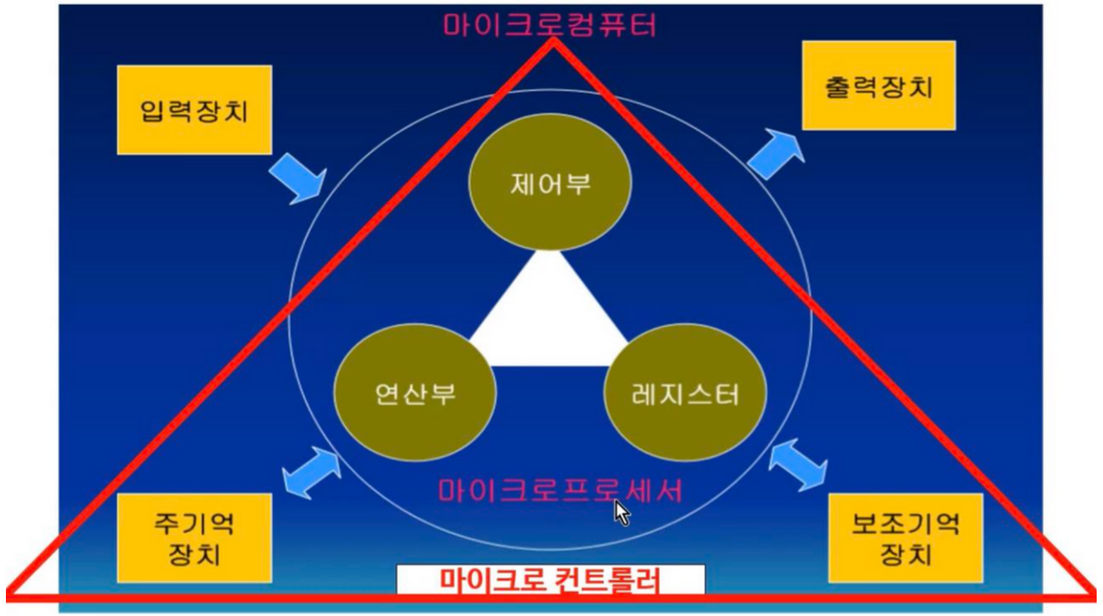
손정웅선생님

OVERVIEW

- 마이크로프로세서 VS 마이크로컨트롤러 VS 마이크로컴퓨터
- 마이크로프로세서
- 임베디드 시스템
- WSL & Ubuntu 20.04 설치, 세팅
- 리눅스
- VSCode, WSL
- C 프로그램 생성 과정
- gcc
- 포인터, 배열

마이크로프로세서 VS 마이크로컨트롤러 VS 마이크로컴퓨터

Microprocessor VS Microcontroller VS Microcomputer



- 마이크로프로세서(MicroProcess)
  : 컴퓨터 중앙 처리 장치(CPU)의 핵심 기능을 통합한 집적 회로(IC)
- 마이크로컨트롤러 (MicroController Unit, MCU)
  : 마이크로프로세서에 연결 되는 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 주 변장치들이 연결되어 동작하는 것이고, 마이크로컨트롤러는 CPU 내부에 메모리와 주변장치들이 포함되어 있어 단독 동작이 가능

마이크로프로세서

Microprocessor

특징 :

- 저비용: 집적 회로 기술로 저렴한 비용으로 이용가능
  -> 컴퓨터 시스템의 비용 감소 가능
- 고속: 초당 수백만개의 명령을 실행, 빠른 속도로 작동
- 작은 사이즈: 전체 컴퓨터 시스템의 크기 감소 가능
- 다재다능: 용도가 매우 다양, 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 -> 휴대 가능

종류:

- 8051
  - x86 CPU 생산업체인 인텔(Intel)에서 만든 MCU
  - TI사의 TMS1000 MCU와 더불어 1975년 개발, 초창기 MCU
- PIC(Peripheral Interface Controller)
  - MicroChip Technology사에서 만든 MCU
  - 산업용으로 많이 사용, 주변의 가전제품 多
- AVR(Advanced Virtual RISC)
  - Atmel사에서 만든 MCU
- 개발 환경이 잘 구성, 새로운 프로그램 기록 방법 용이
- PIC보다 처리 속도가 빠르며, 8051보다 학습자료 풍부

마이크로프로세서

Microprocessor

종류 :

- ARM(Advanced RISC Machine)

 - 저전력의 MCU로서 거희 PC에 버금가는 활용도를 자랑

 - OS 탑재 가능, 많은 제조사에서 ARM 회로도 사용한 MCU 제작

 - 기존의 다른 MCU와 다른점: ARM은 회로설계만 하고 제조 X

RISC / CISC 개념

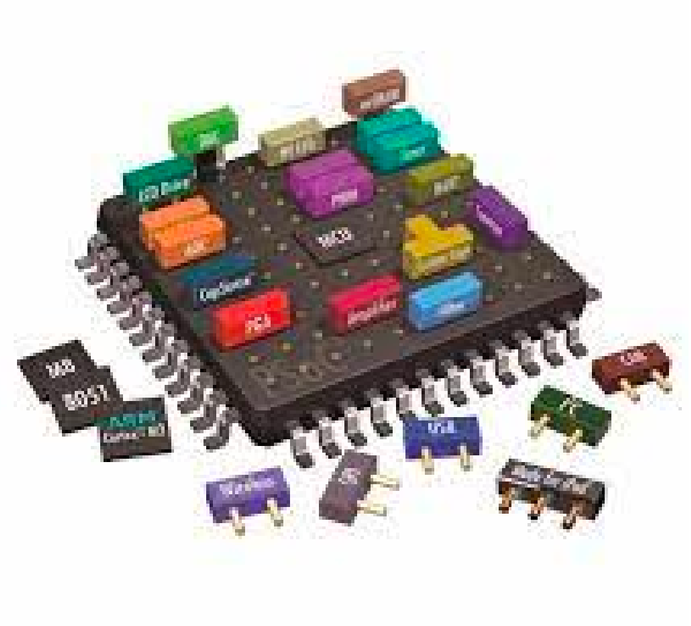
- CPU(중앙처리장치) 를 설계하는 방식
- CPU가 작동하려면 프로그램 + 명령어를 주입해서 설계
- RISC : 명령어가 H/W적인 방식
- CISC : 명령어가 S/W적인 방식

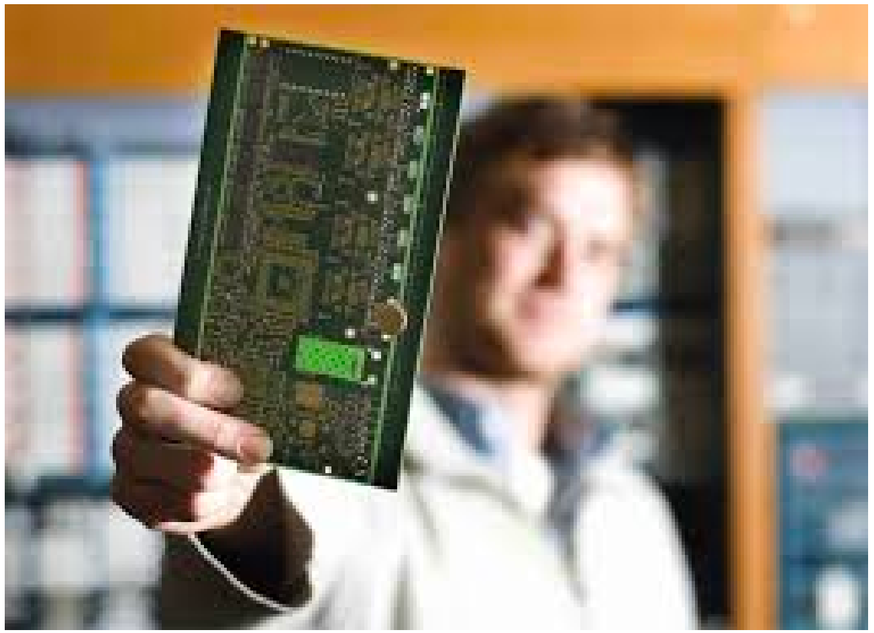
분류	CISC	RISC
설계	복잡	단순
전력소모	많음	적음
처리속도	느림	빠름
명령어 종류	많음	적음
명령어 길이	가변길이 명령어	고정길이 명령어
프로그래밍	간단	복잡
레지스터	적음	많음
제어방식	마이크로 프로그래밍 방식	하드와이어드 방식
용도	PC용 프로세서	워크스테이션

임베디드 시스템

Embedded system

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





임베디드 시스템

Embedded system

[그림 1] 전기밥솥은 알기 쉬운 임베디드 시스템

[그림 5] 키보드 입력도 화면 표시도 없는 임베디드 기기에서 어떻게 프로그램을 개발하나?

WSL & Ubuntu 20.04 설치, 세팅

WSL & Ubuntu 20.04 installation, setting

- 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>
- 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

리눅스_특징

Linux_features

- 이식성, 확장성 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

Linux_Command

- 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..:한단계상위폴더로이동
- 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
- 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(빈폴더가 아닐 경우 삭제 안됨)
- rm -rf [폴더명] : 비어있지 않은 폴더 삭제
- [명령어] --help : 명령어 도우미
- $ 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

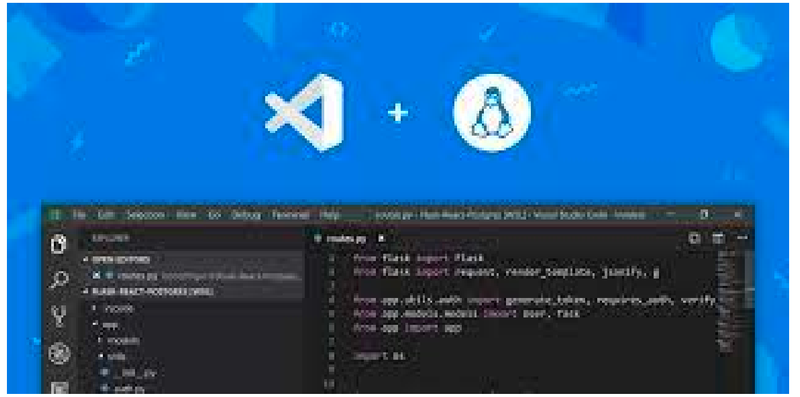
VSCode, WSL

VSCode, WSL install

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

- extension tab에서 Remote-WSL 추가 설치



- Ubuntu에서 VSCode 실행
- $ code . // VSCode 실행
- Ubuntu상에서 vscode 실행한 상태
  (좌측 하단에 WSL:Ubuntu-20.04 표시)

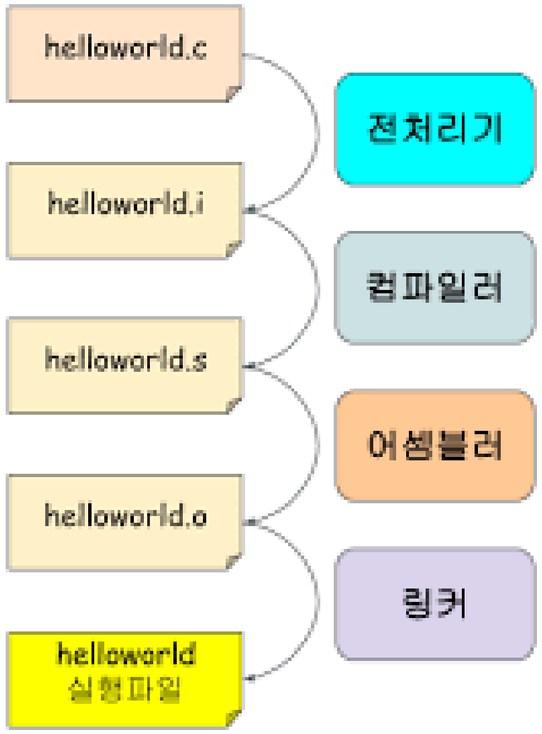
Portable Mode에서 VSCode 자동실행

Autorun VSCode in Portable Mode

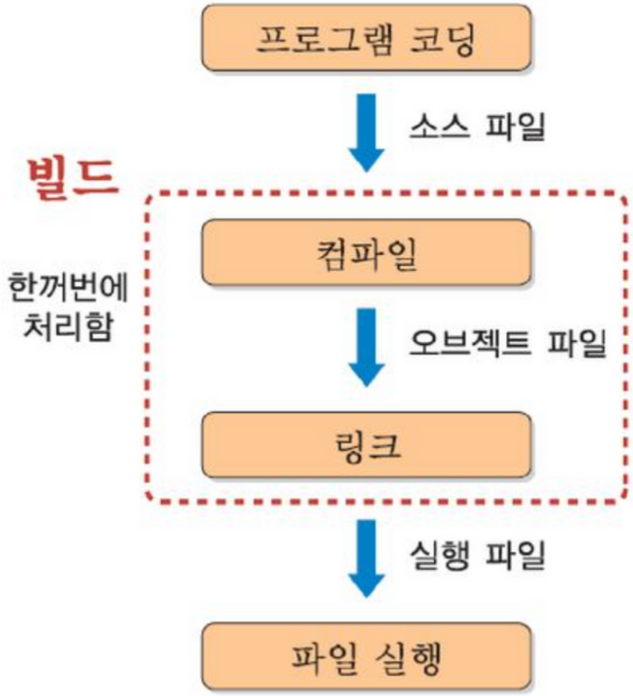
- Ubuntu에서 아래 경로를 입력하면 VSCode 실행
  $ /mnt/c/VSCode/bin/code .
  ※ 참고) 윈도우 경로 : c/VSCode/bin/code
- Ubuntu에서 VSCode가 자동 실행되도록 환경 변수 등록 (반드시 home 디렉토리에서 실행 : cd ~)
- $ vi .bashrc // vi편집기를 이용해서 PATH 수정
- 리눅스 환경변수에 등록하기 위해 bashrc에 PATH 기입 후 저장 (참고) 전체 환경 변수 확인 : $ echo $PATH
- .bashrc에서 맨 아래 다음의 PATH 설정 추가 export PATH=$PATH:/mnt/c/VSCode/bin/
- ※ 다른 셸(예 .zshrc)을 사용할 경우에는 반드시 추가 PATH 설정할 것
- Ubuntu에서 다음의 명령 입력
- $ source .bashrc // 실행된 환경에서 다시 실행(리부팅하지 않아도 됨)
- $ tail -n1 ~/.bashrc // 셸의 마지막에 수정된 내용 확인
- $ code . // VSCode 실행

C 프로그램 생성 과정

C program creation process



- C 프로그램의 작성과 실행 순서



소스 프로그램을 코딩하고, 확장자를 *.c로 한다.

작성된 *.c를 기계어로 변환한다.
그 결과로 오브젝트 파일이 생성된다.

관련 파일을 하나로 결합한다.
그 결과로 실행 파일이 생성된다.

생성된 *.exe 파일을 실행한다.

gcc

gcc

- 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

pointer

포인터 변수 선언 코드

```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;
}
```

역참조1 코드

```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
```

역참조2 코드

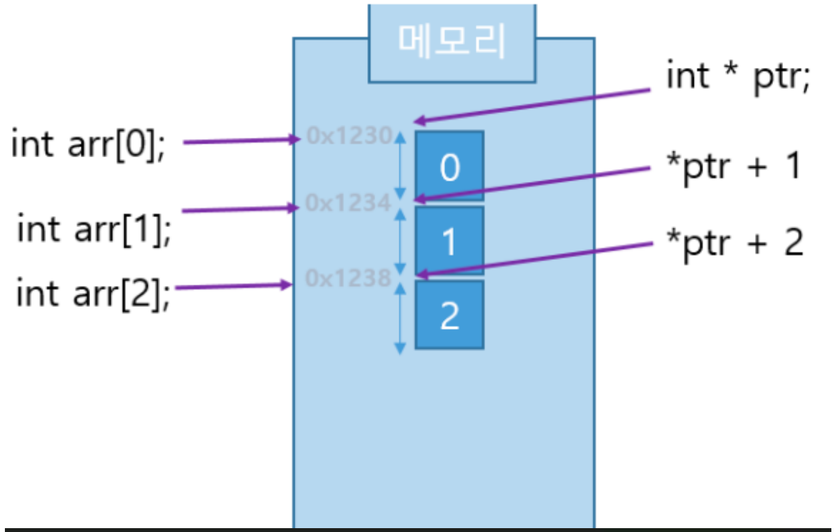
```c
#include <stdio.h>
int main(){
    int *ptr; // 포인터 변수 선언
    int num = 10; // int형 변수를 선언하고 10 저장
    ptr = &num; // ptr에 num의 메모리 주소 저장
    *ptr = 20; // 역참조 연산자로 메모리 주소에 접근하여 20을 저장
    printf("%d\n", *ptr); // 역참조 연산자로 메모리 주소에 접근하여 값 20을 출력
    printf("%d\n", num); // 실제 num의 값도 20으로 바뀌어 출력
    return 0;
}
// 참고) 컴파일 : gcc -o pointer2-1 pointer2-1.c
```

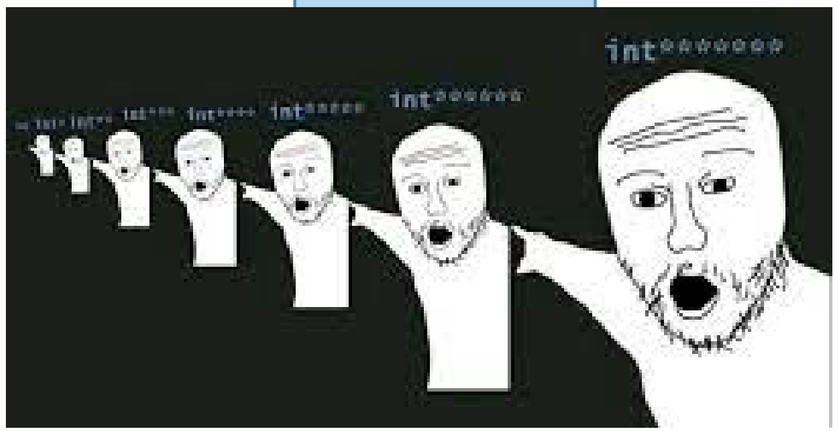
이중포인터 코드

```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;
}
```

배열과 포인터

Arrays and Pointers





- int * ptr = &arr[0] ; 이라고 할 때
  arr의 첫번째 주소를 ptr으로 가리키는 것
- arr 자체가 주소값이므로 int * ptr = arr 으로 도 사용 가능
- *ptr 은 배열의 첫번째 항의 값을 출력함으로 0을 리턴
- *(ptr + 1) 은 ptr의 주소값에서 +1 (int형이므 로 4byte)
  뒤로 더한곳으로 주소를 가리킴

배열과 포인터

Arrays and Pointers

배열과 포인터2 코드

```c
#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; // 첫 번째 값 변경
    *(ptr + 1) = 30; // 두 번째 값 변경, 자료형이 int이므로 1씩 증가하면 4byte씩 이동
    *(ptr + 2) = 300; //세 번째 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;
}
```

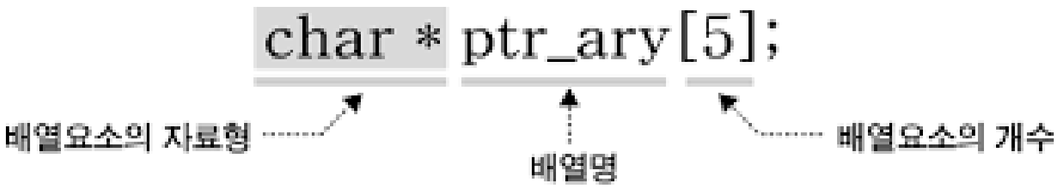
배열과 포인터3 코드

```c
#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; // 첫 번째 값 변경
    ptr[1] = 70; // 두 번째 값 변경, 자료형이 int이므로 1씩 증가하면 4byte씩 이동
    ptr[2] = 100; // 세 번째 값 변경
    printf("변경 후 배열 출력\n"); // 배열 출력 ㄱㄱ
    printf("arr[0] : %d\n", arr[0]);
    printf("arr[1] : %d\n", arr[1]);
    printf("arr[2] : %d\n", arr[2]);
    return 0;
}
```

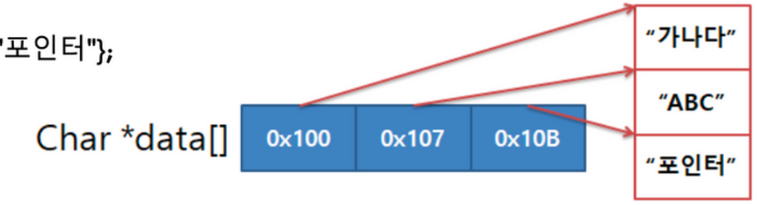
선언한 포인터가 배 열의 첫번째 칸을 가리키면,
포인터도 마치 배열처럼 사용 가능

배열과 포인터

Arrays and Pointers




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



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

배열과 포인터

Arrays and Pointers

포인터 배열2 코드

```c
#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]는 -> 문자열 주소를 가리킴
    for (i = 0; i < 3; i++) {
        printf("arr[%d] -> %s\n", i, arr[i]);
    }
    return 0;
}
```