**<함수 오버로딩>**

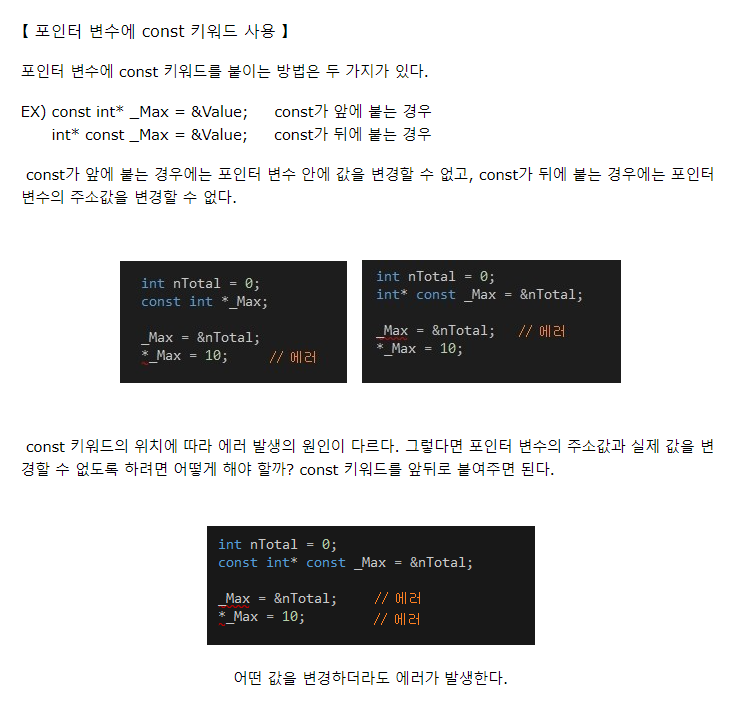
함수 호출 시 전달되는 인자를 통해서 호출하고자 하는 함수를 구분함

C에서는 함수의 이름만 가지고 검색하지만 C++에서는 매개변수의 선언도 따짐

따라서 함수 오버로딩이 가능하려면 매개변수의 자료형 또는 갯수가 달라야함

※ 함수의 자료형이 다른건 안됌, 자료형은 호출 함수를 구분하는 기준이 될 수 없음

**<포인터와 const>**



<https://hh4131.tistory.com/64>

**<프로그램 메모리 영역>**

****

[**https://jinshine.github.io/2018/05/17/%EC%BB%B4%ED%93%A8%ED%84%B0%20%EA%B8%B0%EC%B4%88/%EB%A9%94%EB%AA%A8%EB%A6%AC%EA%B5%AC%EC%A1%B0/**](https://jinshine.github.io/2018/05/17/%EC%BB%B4%ED%93%A8%ED%84%B0%20%EA%B8%B0%EC%B4%88/%EB%A9%94%EB%AA%A8%EB%A6%AC%EA%B5%AC%EC%A1%B0/)

**프로세서(Processor)에서 접근 속도가 빠른 기억장치를 순서대로 나열**

1. 레지스터(Register)  
2. 캐시 메모리(Cache Memory)  
3. 메인 메모리(Main Memory)  
4. 보조 기억장치(하드디스크, CD 드라이브 등)

**레지스터 (Register)**

레지스터는 CPU Core 내부에 존재하며, Core에서 연산을 수행할 때 직접 참조할 수 있는 유일한 기억장치입니다.

레지스터는 Core 내부에도 몇 개 존재하지 않는 귀한 기억 장치이며, 보통 한 워드(Word; 프로세서가 처리할 수 있는 가장 자연스러운 데이터의 크기이며, 보통 4Byte=32Bit로 구성된다.)가 레지스터 하나의 크기가 됩니다.

레지스터들은 각각 사용되는 용도가 정해져 있으며, 유사한 기능을 하는 레지스터들끼리 모아서 레지스터 파일(Register File)을 구성합니다.

**캐시 메모리 (Cache Memory)**

캐시 메모리도 레지스터와 마찬가지로 CPU 내부에 존재하며, Core와 입출력 버스를 통해 정보를 주고받습니다.

보통 명령어를 저장하는 Instruction Cache와 데이터를 저장하는 Data Cache로 구분되어 있으며, 때로는 캐시 메모리도 L1 Cache, L2 Cache 등 여러 레벨으로 분할해서 세부 계층화하기도 합니다.

캐시 메모리는 주로 Refresh(메모리의 내용을 보존하기 위해서 주기적으로 데이터를 갱신하는 작업. 이 작업이 수행중일 때는 메모리에 읽거나 쓰는 동작을 할 수 없다.)가 필요 없는 SRAM(Static RAM)으로 구성합니다.

캐시 메모리는 CPU내부에서 상당한 공간을 차지할 정도로 중요한 기억 소자입니다.

**메인 메모리 (Main Memory)**

메인 메모리(Main Memory)는 CPU 외부에 존재하며, CPU와 North Bridge를 통한 데이터 버스를 통해 통신합니다. 메인 메모리는 데이터를 보존하기 위해 주기적으로 Refresh가 필요한 DRAM으로 구성됩니다.

**보조 기억장치**

보조 기억장치는 하드 디스크나 CD, DVD, USB메모리 등을 포함하며, 메인 메모리에 비해 접근 속도가 현저하게 떨어집니다.

이 외에도 더 하위 레벨에 네트워크 리소스(Network Resource)를 포함시킬 수도 있습니다. 네트워크 리소스의 경우 접근하기 위해서는 원격지에 있는 데이터를 통신회선을 통해 읽어와야 하기 때문에 기억 장치의 계층 중에서 접근 속도가 가장 느립니다.

문제의 핵심은, 하위 계층 메모리로 갈 수록 접근 속도가 떨어진다는 점입니다.

요즘 출시되는 프로세서들은 보통 수 GHz의 속도로 동작하도록 설계되어 있습니다. 이는 Processor에서 1초에 수행하는 연산의 갯수를 의미하기도 하며, 이와 비슷한 속도로 연산에 필요한 레지스터를 접근한다는 것을 뜻합니다.

만약 메모리가 계층화되어 있지 않다면 Processor에서 필요한 데이터를 사용하기 위해 많은 지연시간이 필요하게 되고, 이는 결국 전체적인 성능 하락으로 이어지게 됩니다.

CPU Core에서 CD에 저장된 데이터를 가지고 덧셈(Add) 연산을 수행하려고 하는데, 이놈의 컴퓨터는 CD부터 돌리고 수 초 이상을 아무 일도 하지 않은 채 데이터가 올 때까지 마냥 세월아 네월아 하며 놀고 있다면 사용자의 입장에서는 환장할 노릇일 것입니다.

Processor와 메모리의 접근 속도 차에 따른 성능 하락은 메모리를 계층화하여 Processor에서 접근 속도가 빠른 계층의 메모리에 상대적으로 느린 하위 계층의 데이터를 미리 가져다 놓음으로써 해결할 수 있습니다. 이렇게 데이터를 미리 가져다 놓는 작업을 캐싱(Caching)이라 합니다.

그렇다면 하위 기억계층보다 그 크기가 현저하게 작은 상위 계층에 저장할 데이터는 어떻게 선별해 내는 것일까요?

이는 프로세서의 성능 향상에 있어서 매우 중요한 주제입니다.

메모리를 계층화하더라도 상위 계층의 메모리에 Processor에서 필요로 하는 데이터를 시기적절하게 미리 가져다놓지 않으면 이는 있으나 마나 한 것이 되기 때문입니다.

이 문제를 해결하는 데에는 다음 두 가지의 원리가 적용됩니다.

시간적 지역성(Temporal Locality) : 최근 접근된 내용이 또 다시 접근될 확률이 높다.

공간적 지역성(Spatial Locality) : 한 지역의 내용이 접근되었다면, 그 주위의 내용도 접근될 확률이 높다.

실제로 CPU 내부에 있는 캐시 컨트롤러(Cache Controller)는 이 두 가지 원리를 적용하여 캐시 메모리에 적재될 내용을 관리합니다.

새로운 내용을 읽어들여서 캐시메모리에 적재하려고 하는데 여유 공간이 없다면 가장 오래된 데이터를 삭제하고 적재를 수행하여 시간적 지역성의 원리를 적용하고, 한 번 적재를 시도할 때 Block(Caching과정에서 적재되는 단위 데이터) 단위로 데이터를 가져와서 공간적 지역성의 원리를 적용하는 것입니다.