메모리 구조 (Memory Structure)



[Test]

코드를 실행하기 위해 저장되어 있는 영역(코드 영역)

프로그램을 실행시키기 위해 구성되는 것들이 저장되어 있는 영역 (제어문, 함수, 상수)

[Data]

전역변수, 정적변수 등이 저장되는 공간

메인 함수 전에 선언되어 프로그램이 끝날 때까지 메모리에 남아있는 변수가 특징

초기화된 변수 영역(BSS), 초기화 안된 변수 영역

[Heap]

사용자에 의해 관리되는 영역, 매개변수들이 저되는 공간

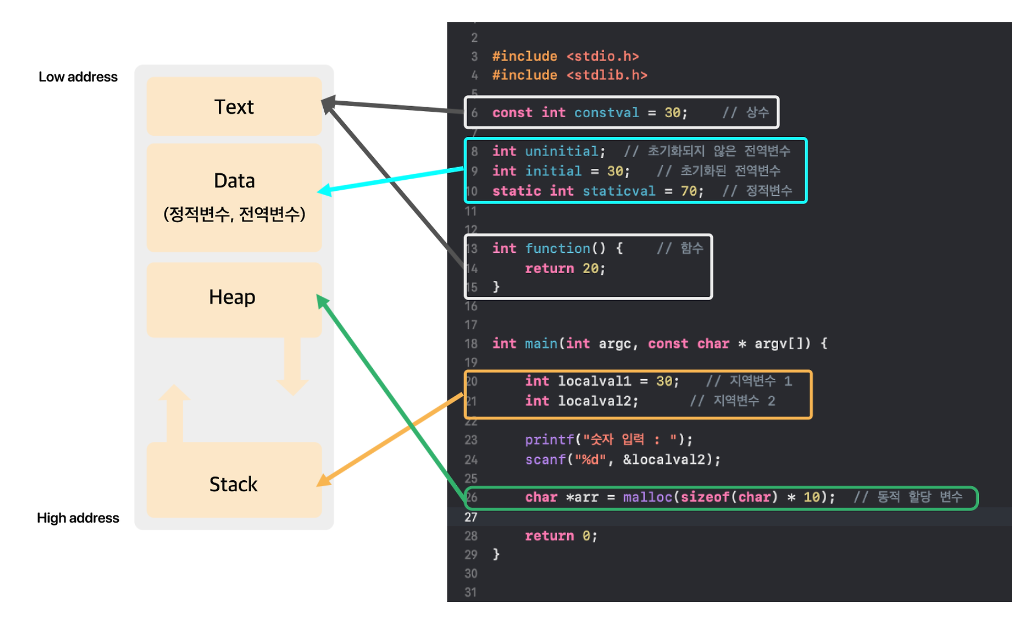
메인 함수안에서의 변수들도 포함, 함수가 종료되면 해당 함수에 할당된 변수들을 메모리에 해제

Stack영역은 Heap영역과 반대로 높은 주소에서 낮은 주소로 이동

메모리 주소(Memory Address)

32bit(4Byte 길이의 주소를 가짐)와 64bit OS (8Byte 길이의 주소를 가짐)

|  |
| --- |
|  |



상수, 함수 Test영역, 전역, 정적변수는 Data 영역, 지역변수들은 Stack 동적할당이 되는 변수들은 Heap 영역에 위치하고, Text영역은 가장 낮은 주소, Data영역이 그 다음 주소, Heap영역이 data 영역의 다음 주소, Stack영역이 가장 높은 주소에 위치함

포인터를 사용하여 메모리 주소를 접근 방식은 실제 물리적 주소가 아닌 가상 메모리 주소

실제는 연속적으로 데이터를 할당하면 선형적으로 메모리를 접근하는 것 가지만 실제로 페이징 기법을 통해 접근하게 되기 때문에 실제 메모리에 할당된 영역은 선형적이지 않음.

**버퍼 오버플로우**

**Buffer Overflow**는 메모리의 버퍼를 넘치는 상태를 의미한다.

Stack과 Heap 영역에서 발생할 수 있으며, 인접한 데이터 영역까지 침범할 수 있다.

C언어에서는 strcpy(), gets(), scanf()와 같은 함수들이 데이터의 크기를 검사하지 않기 때문에 Overflow가 발생할 수 있다.

**스택 오버플로우**

**Stack Overflow**는 호출 스택이 할당된 스택 영역 경계선 밖으로 넘어갈 때 발생한다.

재귀 호출에서 자주 발생하며, 무한 호출로 인해 스택에 데이터가 계속 쌓이게 된다.

**힙 오버플로우**

**Heap Overflow**는 힙 영역에서 할당된 영역의 경계선 밖으로 넘어갈 때 발생한다.

매우 큰 데이터를 생성하려고 할 때 발생하며, C언어에서는 malloc()을 사용하여 동적 할당을 한다.

**엔디언의 정의와 개념**

* **엔디언(Endianness)**: 컴퓨터 메모리와 같은 1차원 공간에 여러 개의 연속된 대상을 배열하는 방법을 의미한다.
* **바이트 순서(Byte order)**: 엔디언은 특히 바이트를 배열하는 방법을 설명하는 데 사용된다.
* **종류**:
  + **빅 엔디언(Big-endian)**: 큰 단위의 바이트가 앞에 오는 방식.
  + **리틀 엔디언(Little-endian)**: 작은 단위의 바이트가 앞에 오는 방식.
  + **미들 엔디언(Middle-endian)**: 두 경우에 속하지 않거나 둘을 모두 지원하는 방식.

**엔디언의 유래**

* **어원**: "엔디언"이라는 용어는 조너선 스위프트의 《걸리버 여행기》에서 유래하였다.
  + **소인국 릴리퍼트 이야기**: 달걀을 깰 때 뭉툭한 끝을 먼저 깨는 사람들(빅 엔디언)과 뾰족한 끝을 먼저 깨는 사람들(리틀 엔디언) 간의 격론에서 유래하였다.
* **논란**: 빅 엔디언과 리틀 엔디언 중 어느 것을 사용할지는 상황에 따라 임의적이며, 종종 논란의 대상이 된다.
* **대니 코언**: 1980년에 쓴 "On Holy Wars and a Plea for Peace"라는 글에서 엔디언이라는 용어가 컴퓨터와 관련하여 사용되기 시작하였다.

**바이트 순서의 종류**

* **바이트 순서**: 크게 두 가지로 나눌 수 있다.
  + **빅 엔디언**: 사람이 숫자를 쓰는 방법과 유사하게 큰 단위의 바이트가 앞에 오는 방식이다.
  + **리틀 엔디언**: 작은 단위의 바이트가 앞에 오는 방식으로, PDP-11과 같은 몇몇 아키텍처는 2바이트 단위와 1바이트 단위로 서로 다른 순서를 사용하기도 한다.
* **비교**:
  + **0x1234의 표현**:
    - 빅 엔디언: 12 34
    - 리틀 엔디언: 34 12
  + **0x12345678의 표현**:
    - 빅 엔디언: 12 34 56 78
    - 리틀 엔디언: 78 56 34 12

**빅 엔디언과 리틀 엔디언의 장단점**

* **빅 엔디언의 장점**:
  + 소프트웨어 디버깅이 용이하다.
  + 사람이 숫자를 읽고 쓰는 방식과 유사하여 메모리 값을 쉽게 확인할 수 있다.
* **리틀 엔디언의 장점**:
  + 메모리에 저장된 값의 하위 바이트를 사용할 때 별도의 계산이 필요 없다.
  + 예를 들어, 32비트 숫자 0x2A는 리틀 엔디언으로 표현하면 2A 00 00 00이 되어 하위 비트를 쉽게 얻을 수 있다.
* **가산기 설계**:
  + 리틀 엔디언에서는 LSB(Least Significant Bit)부터 시작하여 덧셈을 수행하므로 설계가 단순해진다.
  + 빅 엔디언에서는 마지막 바이트부터 시작하여 역방향으로 진행해야 한다.

**바이 엔디언 아키텍처**

* **바이 엔디언(Bi-endian)**: 몇몇 아키텍처는 빅 엔디언과 리틀 엔디언 중 하나를 선택할 수 있도록 설계되어 있다.
* **대표적인 아키텍처**:
  + ARM, PowerPC, DEC 알파, MIPS, PA-RISC, IA-64 등이 있다.
* **바이트 순서 변경**: 대부분의 경우 컴퓨터가 시작된 상태에서 소프트웨어적으로 바이트 순서를 바꿀 수 있지만, 일부는 하드웨어에 내장된 펌웨어에서 선택해야 할 수도 있다.

**미들 엔디언의 개념**

* **미들 엔디언(Middle-endian)**: 한 방향으로 순서가 정해져 있지 않은 경우를 의미한다.
  + 예를 들어, 32비트 정수가 2바이트 단위로는 빅 엔디언이고, 1바이트 단위로는 리틀 엔디언인 경우가 있다.
* **사용 예**:
  + VAX와 ARM에서는 배정밀도 부동소수점 실수를 미들 엔디언으로 저장한다.
  + PDP-11에서는 32비트 정수를 미들 엔디언으로 저장하며, 이를 PDP 엔디언이라고 부르기도 한다.

**비트 순서의 중요성**

* **비트 접근**: 바이트나 옥텟은 원자적인 단위로 간주되지만, 비트 단위의 접근이 필요할 수 있다.
* **비트 순서**: 비트 순서는 상대적으로 덜 중요하지만, 저수준에서 비트 단위의 데이터를 최상위 비트부터 채울 것인지 최하위 비트부터 채울 것인지에 대한 문제가 발생할 수 있다.
* **C 언어의 비트 필드**: C는 구조체에서 바이트보다 더 작은 단위의 변수를 선언할 수 있는 비트 필드를 지원한다.
  + **최상위 비트 우선**: 최상위 비트(MSB)부터 채우는 방식.
  + **최하위 비트 우선**: 최하위 비트(LSB)부터 채우는 방식.
* **파일 포맷 예시**: PNG와 GIF는 각각 최상위 비트 우선과 최하위 비트 우선을 사용하는 대표적인 파일 포맷이다.