

고려대학교 ALPS 2021

C언어 스터디



Week 0x05

메모리, 동적 할당

Contents

0x01 동적 메모리 할당 Dynamic Memory Allocation

0x02 문자열 처리 함수 String Processing Functions

0x03 실습 및 과제 Practice & Assignments

0x01 Dynamic Memory Allocation

동적 메모리 할당

오늘은 동적 메모리 할당에 대해서 배워보도록 하겠습니다.
우리가 코드에서 사용하는 변수들은 모두 메모리에 저장되게 됩니다.

동적 메모리 할당을 배워보기 전에 먼저 메모리에 대해 알아보시다.
메모리는 코드 Code, 데이터 Data, 스택 Stack, 힙 Heap으로 나눌 수 있습니다.

먼저 Code는 말 그대로, 우리가 작성한 코드가 들어가는 부분입니다.
작성된 코드를 실행하게 되므로 프로그램이 돌아가는 동안
계속해서 메모리 안에 있게 되고,
CPU는 이 코드 영역의 코드들을 하나씩 가져가서 지시대로 처리하며 실행됩니다.

0x01 Dynamic Memory Allocation

동적 메모리 할당

data 영역은 프로그램의 전역 변수, static 변수가 저장되는 영역입니다.

전역 변수는 프로그램 전체에서 쓰이는 변수이고,
static 변수 역시 한 번만 초기화되는 변수로
프로그램 전역에 영향을 미치기 때문에 프로그램 시작 시 할당되어
프로그램이 끝날 때 까지 계속 메모리에 들어있습니다.

0x01 Dynamic Memory Allocation

동적 메모리 할당

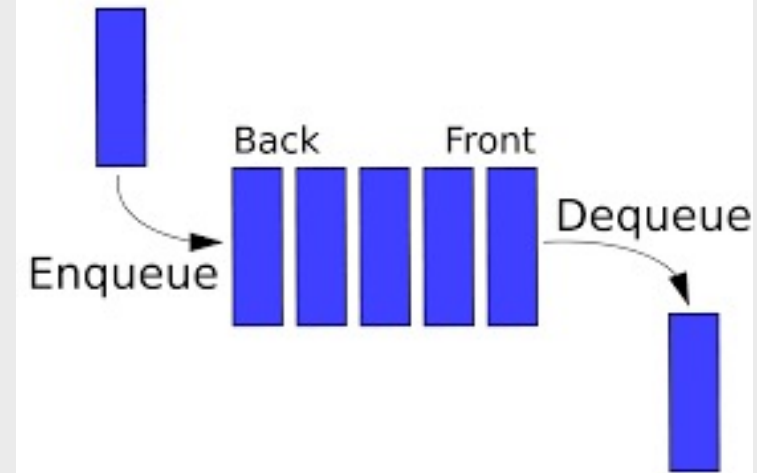
다음은 stack입니다.

stack이라는 자료구조에 대해서 들어보신 적이 있나요?

스택, 큐, 트리, 힙 등이 여러분이 자료구조 시간에 배울 가장 기본적인 자료구조들입니다. 프로그램에서 데이터를 다룰 때 필수적인 구조들인데,

간단하게 큐와 스택에 대해서만 알아보자면

큐는 '게임 큐를 잡다' 등의 표현의 그 큐 입니다.
먼저 들어간 사람이 먼저 나오게 되는,
즉 먼저 들어간 데이터가 먼저 나오는
FIFO (First In, First Out) 식의 구조입니다.



0x01 Dynamic Memory Allocation

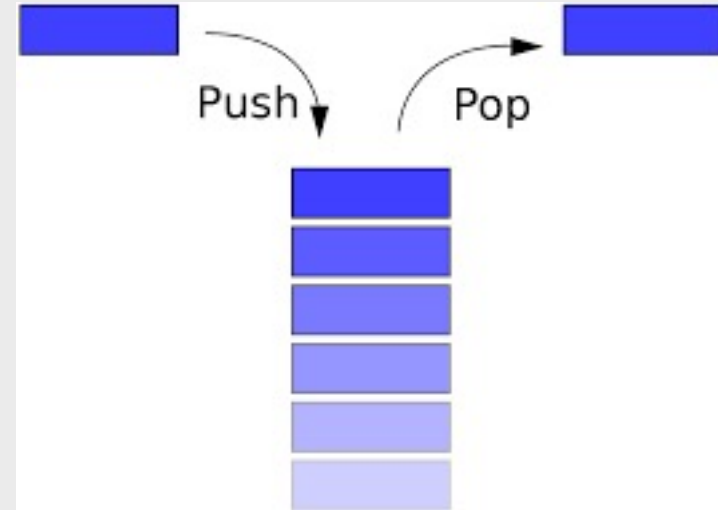
동적 메모리 할당

stack에 대해서 알아보자면 stack은 쌓다라는 의미죠.

쌓인 책을 생각해보면, 책은 위에서부터 쌓을 수 있고, 위에서부터 꺼낼 수 있습니다.

즉, 마지막에 쌓은 책을 먼저 꺼내게 됩니다.
그렇기에 스택은 마지막에 넣은 데이터가 먼저 나오게 되는 LIFO (Last In, First Out) 식의 구조입니다.

굳이 우리가 논하는 범위를 벗어나는 스택과 큐에 대한 설명을 드린 이유는, C언어가 돌아가는 구조를 더 명확히 이해하고자 함입니다.



0x01 Dynamic Memory Allocation

동적 메모리 할당

이제는 정말 스택 메모리에 대해 알아보시다.

스택 메모리는 지역 변수와 매개변수가 저장되는 영역입니다.

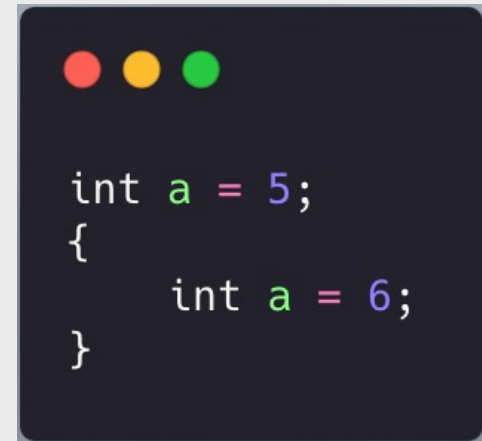
즉, 함수나 블록 안에서 선언되는 변수들을 저장하고 있는 것입니다.
스택의 형식을 이용하여 변수를 저장하고 있기 때문에,

우리가 시험 전에 배웠던, 변수 가림 등의 일이 벌어질 수 있습니다.

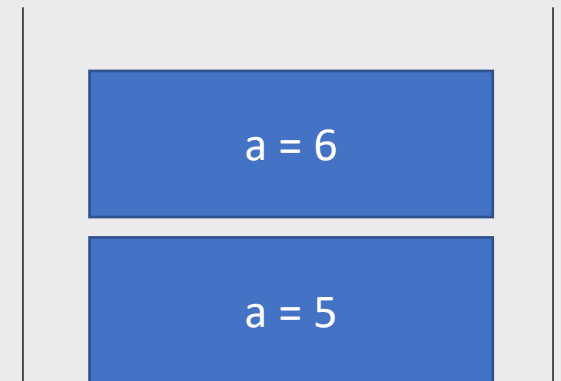
오른쪽 코드를 보면, a=5라는 변수를 선언하고,
다른 블록 안에서 a=6 변수를 선언하면 블록 안에서는 a=5라는
변수가 가려지고 대신 a=6이 사용된다고 했었죠.

이것이 바로 스택의 구조 때문에 발생하는 일이라고 할 수 있습니다.
나중에 들어온 것이 먼저 사용되고, 블록을 벗어나면,
a=6이라는 저 변수는 스택에서 다시 사라지고
a=5가 사용되는 것이죠.

함수의 매개변수 등도 저장하고 있기에, 함수에서도 똑같이 생각하시면 됩니다.



```
int a = 5;
{
    int a = 6;
}
```



0x01 Dynamic Memory Allocation

동적 메모리 할당

그럼 이제 stack과 비교하는 방식으로 heap 메모리에 대해서 알아보시다.

스택 메모리는 프로그램이 자동으로 저번 시간에 배운 변수의 수명에 따라 메모리를 관리해 줍니다.

그에 반해 힙 메모리는 프로그래머가 직접 메모리를 할당하고 해제하며 관리해줘야 합니다.

스택 메모리와 힙 메모리는 같은 공간을 공유하기에 그 크기가 반비례이며

스택 메모리는 메모리의 높은 주소에서 낮은 주소 방향으로 할당되고, 힙 메모리는 메모리의 낮은 주소에서 높은 주소 방향으로 할당됩니다. (포인터 주소)

또한 스택 메모리는 컴파일 시 그 크기가 결정되며, 힙 메모리는 프로그램이 돌아가면서, 즉 런타임에 그 크기가 결정됩니다.



0x01 Dynamic Memory Allocation

동적 메모리 할당

이상 컴퓨터의 메모리에 대해 알아보았는데,
이제 동적 할당을 왜 해야할지에 대해 알아보시다.

오른쪽 코드를 봅시다.

지금까지 배운대로 생각해보면, 배열은 스택 영역에 저장될 것입니다.
그런데, 스택 영역의 메모리는 컴파일 시 결정된다고 했습니다.

컴파일 시, MAX의 크기가 얼마인지 컴파일러는 알 수 없기에 컴파일 시 배열 arr에 얼마만큼의
메모리를 할당할 지 알 수 없기에, 에러를 발생시킵니다.

비주얼 스튜디오 환경에서는 아마 에러가 발생할 것입니다.

그러나, 맥 환경을 사용하시는 등 다른 컴파일러를 사용하시는 분들은 이 코드가 동작이 되실겁니다.

일부 컴파일러에서는 VLA (Variable Length Array), 가변 길이 배열이 가능하기에 그렇습니다.
코드에 따라 배열의 길이를 결정할 수도 있고, 스택이 힙보다 빠르다는 장점이 있으나,

스택의 제한 범위를 넘어가면 오버플로우 문제가 발생할 수도 있는 등의 단점이 있습니다.



```
int main(){  
    int MAX = 100;  
    int arr[MAX];  
}
```

0x01 Dynamic Memory Allocation

동적 메모리 할당

동적 할당을 사용하면 환경의 제한에 구속받지 않고 비주얼 스튜디오에서도 이전과 같이 배열의 크기를 조절할 수 있으며,

아까 잠시 소개드렸던 힙, 스택, 트리 등의 자료구조를 구현할 수 있습니다.

이제는 그 방법을 알아보시다.

<stdlib.h> 라이브러리의 malloc, calloc, realloc, free 함수가 있는데,
이 중 malloc과 free가 동적 할당에서 중요한 역할을 합니다.
각각 할당과 해제에 해당하는 역할입니다.

0x01 Dynamic Memory Allocation

동적 메모리 할당

```
int* ptr = (int*)malloc(sizeof(int));  
int* arr = (int*)malloc(sizeof(int)*MAX);
```

malloc은 Memory ALLOcation의 약자로,
할당할 메모리 바이트 수를 매개변수로 주어 호출하면,
메모리를 힙 영역에 할당한 후 그 메모리 주소의 포인터를 반환합니다.

그래서 위의 코드 예제를 보면, 자료형의 메모리 크기를 반환하는 sizeof 함수를 이용하여
int 자료형의 크기를 인자로 주어, 그 크기인 4바이트 만큼의 메모리를 받아오게 된 것입니다.
받아온 포인터는 자료형이 없기에 ()로 타입 캐스팅을 해주는 것이 좋습니다.

아래의 arr처럼 작성하면, 4바이트 * 배열 길이 만큼 메모리를 할당하여 int*의 타입 형태로 arr에
저장을 했기에, arr은 배열처럼 사용이 가능합니다.

0x01 Dynamic Memory Allocation

동적 메모리 할당

동적으로 할당된 메모리는, 우리가 할당된 메모리를 해제하지 않는 한 계속해서 그 자리를 차지합니다. 그러므로, 메모리 누수 문제가 발생하지 않기 위해서는,

할당한 메모리를 무조건 해제해주어야 합니다.

방법은 아주 쉽습니다.

free 함수에 동적할당된 메모리의 포인터를 인자로 전달하면 됩니다.



```
free(ptr);  
free(arr);
```

free를 한 이후에는 ptr 주소 속의 내용도 사라지고, 더이상 그 주소도 남아있지 않습니다.

free 이후에 다시 해당 메모리에 접근하면, 에러가 발생합니다.

또한, 이미 free 된 메모리를 한번 더 free 하려고 해도 에러가 발생합니다.

이러한 메모리 누수나 잘못된 포인터에 접근하는 에러 해결이 여러분이 자료구조나 알고리즘 수업 과제에서 맞이할 가장 큰 어려움일 겁니다.

단 한번 free해주면 된다고 생각하고 계시면 될 듯합니다.

다른 메모리에 관한 내용들은 다음 시간에 이중 포인터, 구조체와 함께 배워봅시다.

0x02 String Processing Functions

문자열 처리 함수

알고리즘 문제를 풀 때 유용하게 사용할 수 있는 문자열 처리 함수들에 대해서 알아보시다.
해당 함수들은 `string.h` 헤더에 정의되어 있습니다.

문자열 검색 함수

문자열을 받으면 그 중에서 무엇이 있는지 봐야겠죠?

문자열 검색 함수들을 이용하면 간단하게 문자열에서 부분 문자열을 찾을 수 있습니다.

`strchr` 함수는 문자열 중에 있는 문자를 찾아 위치를 반환합니다.

`strrchr` 함수는 문자열 중에 있는 문자를 뒤에서부터 찾아 위치를 반환합니다.

`strstr` 함수는 문자열 중에 있는 부분 문자열을 찾아냅니다.

`strtok` 함수는 문자열을 일정한 delimiter에 따라 잘라주는 역할을 합니다. (tokenize)

`strspn` 함수는 일치하지 않는 첫 문자의 인덱스를 반환합니다.

`strcspn` 함수는 일치하는 첫 문자의 인덱스를 반환합니다.

`strpbrk` 함수는 `strcspn`과 비슷하지만, 일치하는 첫 문자의 위치의 주소값을 반환합니다.

0x02 String Processing Functions

문자열 복사 함수

strcpy나 strncpy를 통해서 문자열을 복사할 수 있게 됩니다.

strcpy 함수는 문자열을 다른 char형 포인터나 배열에 복사합니다.

strncpy 함수는 strcpy 함수에 해당하는 길이를 n개로 제한할 수 있습니다.

문자열 연장 함수

strcat이나 strncat를 이용해서 두 문자열들을 합칠 수 있습니다.

strcat 함수는 두 문자열을 합쳐줍니다.

strncat 함수는 두 문자열을 n 길이만큼만 합쳐줍니다.

0x02 String Processing Functions

문자열 정렬 함수

strcmp나 strncmp를 이용해서 전체나 일부분을 비교할 수 있습니다.
이를 통해 사전식 순서 *Lexicographical order*로 정렬하는 것이 가능합니다.

strcmp 함수는 두 문자열을 비교해줍니다.
strncmp 함수는 두 문자열을 n 길이만큼만 비교합니다.

문자열 초기화 함수

memset을 이용한 배열 초기화를 할 수 있고, strlen을 통해서 문자열의 길이를 잴 수도 있습니다.

memset 함수는 배열을 특정한 문자로 초기화해줍니다.
strlen 함수는 문자열의 길이를 반환합니다.

0x05 Assignment

과제

<https://www.acmicpc.net/problem/1316> - 1316 : 그룹 단어 체커 - 문자열

<https://www.acmicpc.net/problem/1181> - 1181 : 단어 정렬 - 문자열

1문제 이상

이왕이면 저한테 마음껏 질문하시면서 다 풀어보세요!

부족한 수업
들어 주시느라 수고 많으셨습니다!

들어주셔서 감사하고, 질문과 밥약은 언제든지 환영이에요

다음 주에 봅시다~