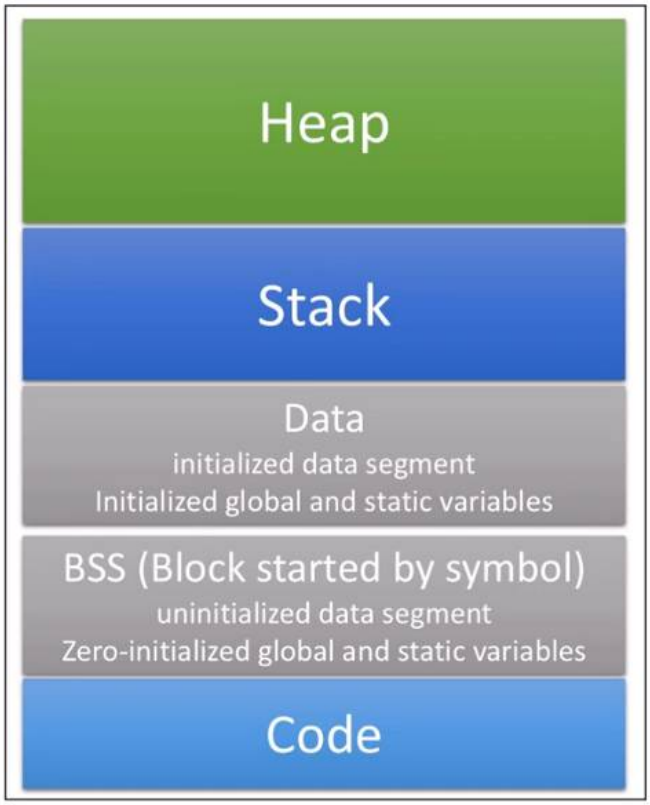
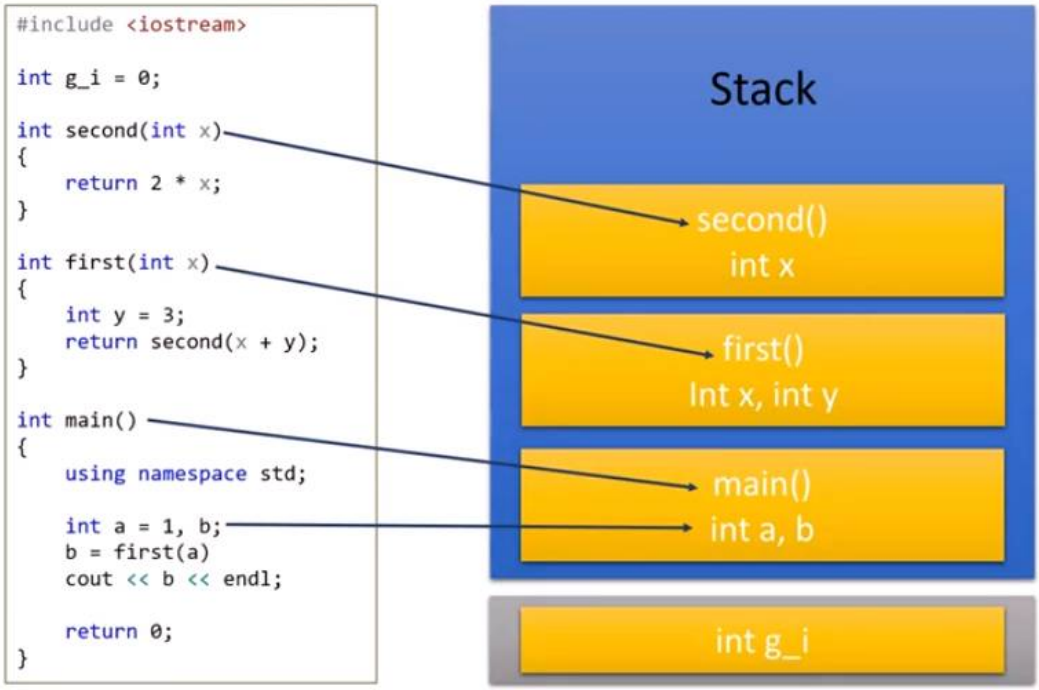


메모리 세그먼트(segment) - 각각의 역할이 다름

- Code segment - 작성한 프로그램 저장
- Data segment(initialized data segment) - 초기화된 global and static variables
- BSS segment(uninitialized data segment) - 0으로 초기화된 global and static variables
- Stack segment - '쌓는다'는 의미
- Heap segment

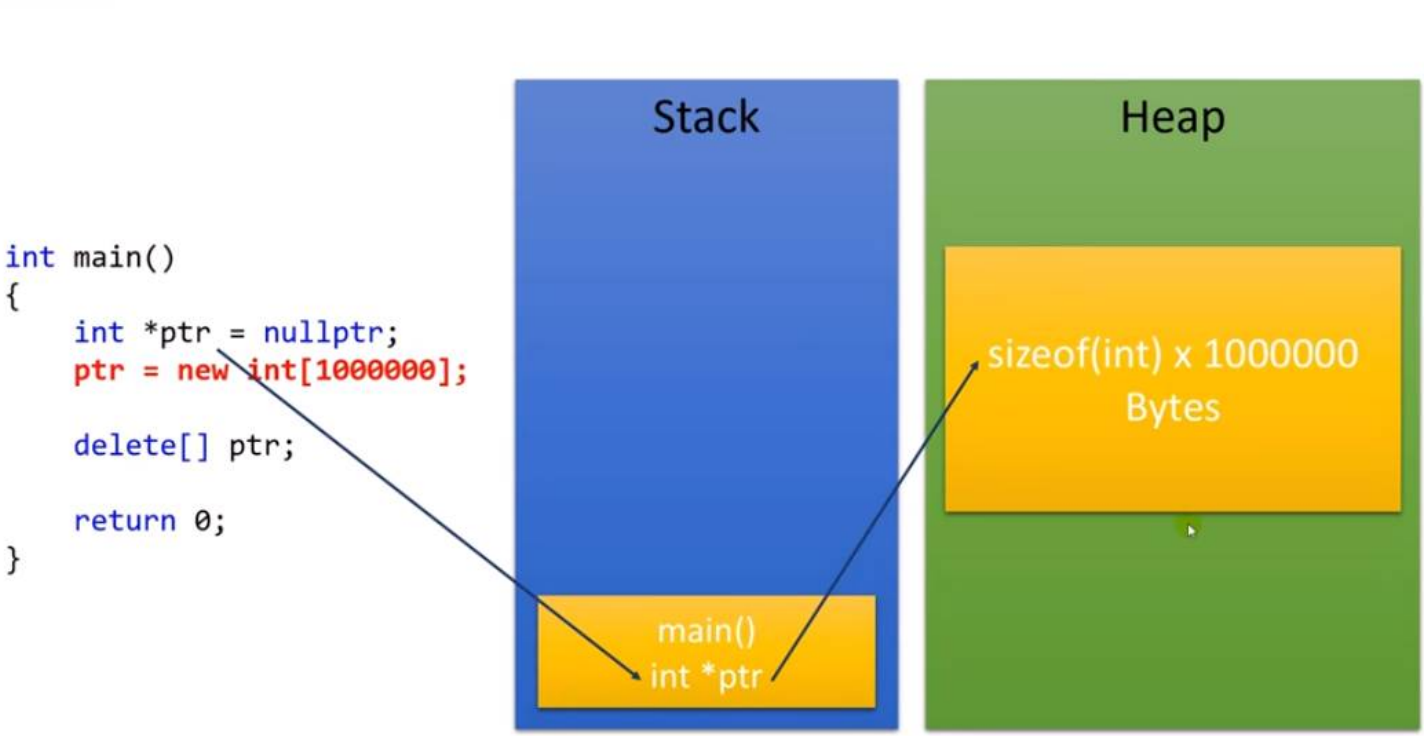


1. 전역 변수가 Data or BSS segment에 자리 잡음
2. main function과 local variables가 stack segment에 쌓임
3. 그 외 function과 function variables가 역순으로 stack segment에 쌓임
4. 실행이 끝나면 위에서부터 사라짐

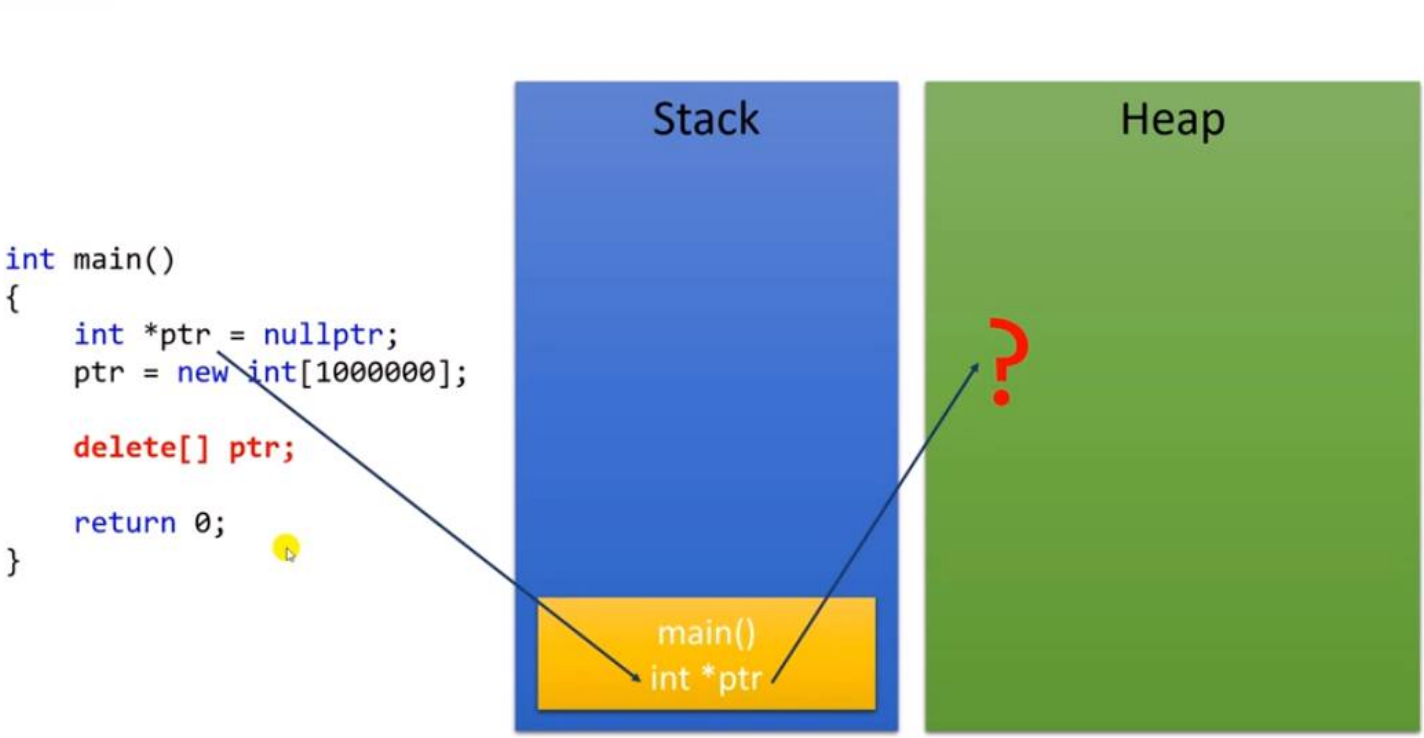
- 비교적 속도가 빠름

```
int main()
{
    // … …

    int array[1000000];

    return 0;
}
```

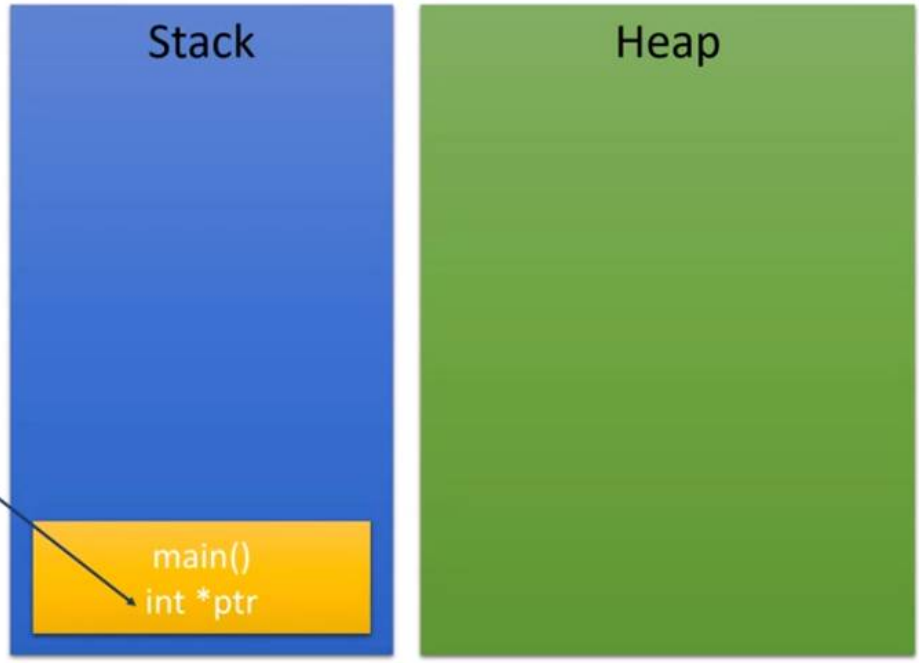
stack의 사이즈의 한계가 있어 메모리 공간이 넘쳐 부족한 경우가 존재



Heap은 사이즈가 크지만 어디에 들어올지 예측하기 힘듬

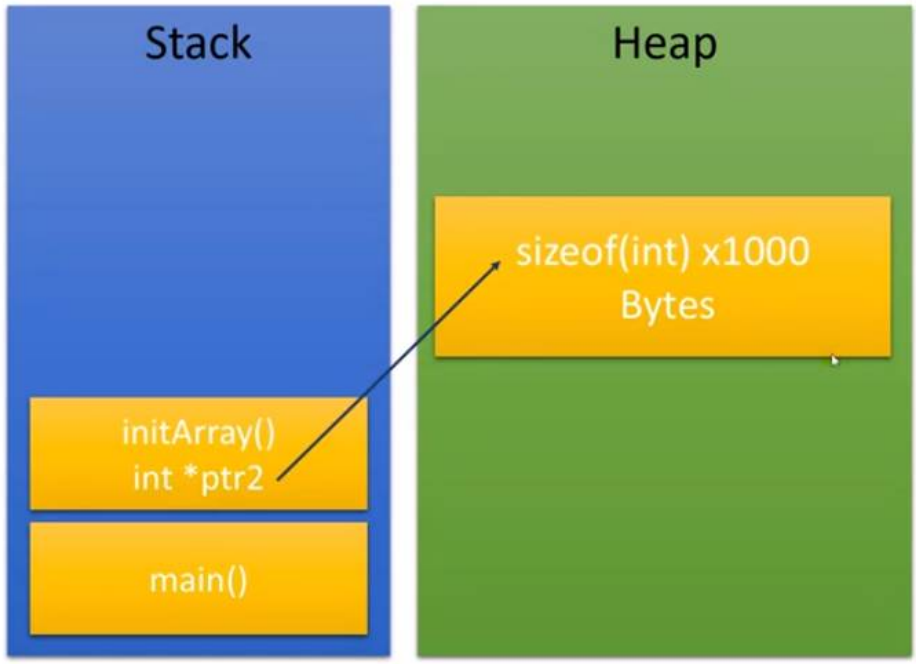


```
int main()
{
    int *ptr = nullptr;
    ptr = new int[1000000];

    delete[] ptr;

    ptr = nullptr;

    // … …

    return 0;
}
```

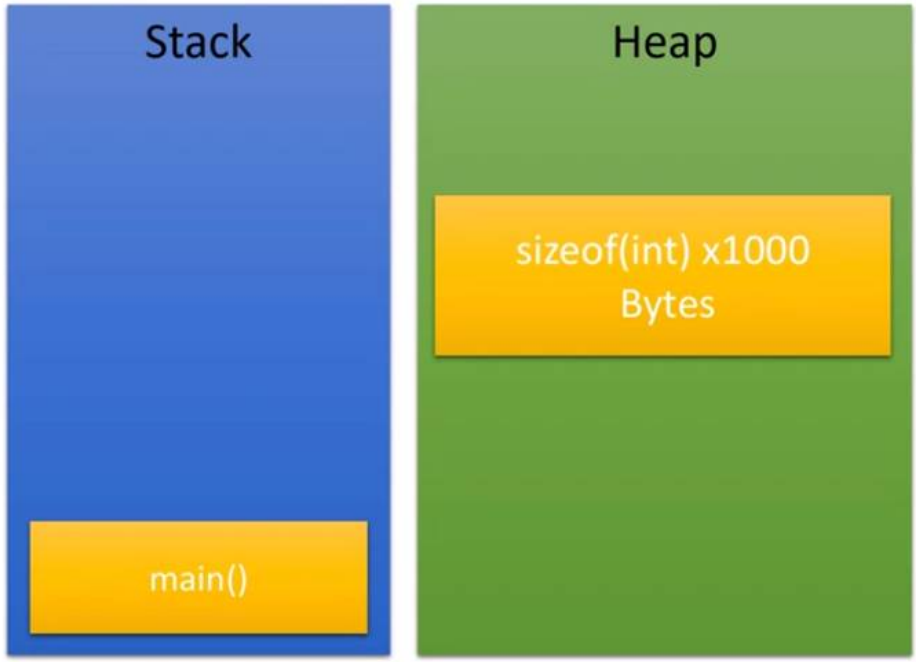


delete을 안 할 경우

```
void initArray()
{
    int *ptr2 = new int[1000];
    // delete [] ptr2;
}

int main()
{
    initArray();

    return 0;
}
```



```
void initArray()
{
    int *ptr2 = new int[1000];
    // delete [] ptr2;
}

int main()
{
    initArray();

    return 0;
}
```



delete을 안하면 Heap에는 할당된 메모리가 계속 존재하고, 이 메모리는 어디 있는지 찾기 힘들기 때문에 사실상 사용하지 못한다.

따라서, 이러한 작업이 반복되면 지속적인 메모리 누수로 프로그램 성능 이 저하 되거나 필요한 메모리를 제대로 사용하지 못해 큰 문제가 된다.