메모리 할당 - new, malloc, etc.

날짜	@2023년 3월 13일

C

malloc

void * maloc(size_t size); *→ 성공 시 할당된 메모리의 주소 값, 실패 시 NULL 반환*

인자로 전달된 정수 값에 해당하는 바이트 크기의 메모리 공간을 힙 영역에 할당하고 이 메모리 공간의 주소값을 **void* 형으로 반환**한다. 이를 *동적 할당(dynamic allocation)*이라고 한다.

따라서 해당 값을 사용하기 위해서는 포인터 형을 변환해야 한다.

```
int * ptr1 = (int *)malloc(sizeof(int));
double * ptr2 = (double *)malloc(sizeof(double));
int * ptr3 = (int *)malloc(sizeof(int)*7);
double* ptr4 = (double*)malloc(sizeof(double)*9);
```

malloc으로 동적 할당 한 메모리는 사용이 끝난 후 반드시 ***free*** 함수를 호출해 해제시켜줘야 한다. 그렇지 않은 경우 메모리 누수가 발생할 수 있다.

free

void free(void * ptr);

realloc

void * realloc(void * ptr, size_t size); → *성공 시 새로 할당된 메모리의 주소 값, 실패 시 NULL 반환*

기존에 malloc으로 할당되었던 메모리 공간을 확장한다.

할당 방법

1. 기존에 할당된 메모리 공간 뒤에 여유가 있다면 ***이어서 확장***
2. 공간이 부족한 경우 ***힙의 새로운 위치***에 메모리 공간을 할당하고 ***값을 복사해 이동***시킨다.

➕ 새로운 메모리에 realloc 하는 경우, 이전 메모리에 대해서 free를 해줘야 하는가?

https://stackoverflow.com/questions/46461236/does-realloc-free-the-existing-memory

⇒ if realloc decided to follow the first approach (i.e. allocate a new memory block), then ***the old block is indeed freed by realloc.***

calloc

void * calloc(size_t elt_count, size_t elt_size); → *성공 시 할당된 메모리의 주소 값, 실패 시 NULL 반환*

malloc과는 달리 블록의 크기와 개수를 인자로 받는다. 또한 malloc과는 달리 메모리 할당과 동시에 메모리 공간을 0으로 초기화한다.

C++

new

C++에서 메모리를 할당하기 위한 연산자. malloc과 달리 반환된 주소의 형 변환을 할 필요가 없다.

new 역할

1. 메모리 공간의 할당
2. 생성자의 호출
3. 할당하고자 하는 자료형에 맞게 반환된 주소 값의 형 변환

new 예외

메모리 공간의 할당이 실패하면 **bad_alloc**이라는 예외가 발생한다.

new 오버로딩

*생성자의 호출, 반환된 주소 값의 형 변환*을 제외한 **메모리 공간의 할당만 오버로딩**이 가능하다. 나머지 두 작업은 C++의 컴파일러가 담당한다.

아래 코드에서 볼 수 있듯이 Point 클래스 내부에 오버로딩 된 new를 Point 클래스를 생성할 때 호출할 수 있는 이유는 ***operator new 함수와 operator delete 함수가 static 함수*** (로 간주되)기 때문이다.

```
class Point
{
private:
    int xpos, ypos;
public:
    Point(int x=0, int y=0):xpos(x), ypos(y) {}
    friend ostream& operator<<(ostream& os, const Point& pos);

    void * operator new (size_t size)
    {
        cout<<"operator new : "<<size<<endl;
        void * adr = new char[size];
        return adr;
    }
    void operator delete(void * char)
    {
      cout<<"operator delete ()"<<endl;
      delete []adr;
    }
};

ostream& operator<<(ostream& os, const Point& pos)
```

```
{
    os<<'['<<pos.xpos<<", "<<pos.ypos<<']'<<endl;
    return os;
}

int main()
{
    Point * ptr = new Point(3, 4);
    cout<<*ptr;
    delete ptr;
    return 0;
}
```

delete

malloc의 free처럼 new를 통해 생성한 객체의 메모리를 해제할 때 사용한다. 메모리가 해제되기 전 소멸자가 먼저 호출되기 때문에 delete는 메모리를 해제하는 역할 만을 담당한다.

문제

Q. malloc과 new의 차이와 각 함수의 장단점은?

malloc	new
realloc으로 메모리 공간 재할당	메모리 공간 재할당 불가능

Q. malloc이 new보다 빠르다고 하는 이유는 무엇인가?

https://stackoverflow.com/questions/2570552/why-are-new-delete-slower-than-malloc-free

⇒ new는 실행 과정에서 생성자를 호출하기 때문에 객체의 크기가 크다면 생성자를 통해 초기화 하는 시간이 길어질 수 있다. 반면 malloc은 메모리 공간을 할당할 뿐 객체를 초기화하지는 않기 때문에 특정 상황에서는 비교적 빠를 수 있다.

https://www.includehelp.com/cpp-tutorial/difference-between-new-and-malloc.aspx

⇒ new는 연산자이고, malloc은 함수이기 때문에 언제나 new가 빠르다.

참고

https://stackoverflow.com/questions/184537/in-what-cases-do-i-use-malloc-and-or-new