C++프로그래밍및실습

Assignment01

Garbage Collector

학과: 인공지능학부

학번: 231583

이름: 유희원

목차

코드 설명 3

실행 결과 9

한계 및 개선사항 11

코드 설명

1. GarbageCollector 클래스
   - 멤버변수

```
#include <vector>
vector<void*> instance;
vector<void**> pointer;
```

     - 벡터를 사용하기 위해 헤더파일을 추가해준다.
     - instance: 새로이 만들어지는 객체의 주소를 저장하기 위한 변수이다. 벡터를 이용하여 관리하고 주소를 저장하기 때문에 void*형이다.
     - pointer: 포인터 변수의 주소를 저장하기 위한 변수이다. 이 또한 벡터를 이용하여 관리하고 이중포인터이기 때문에 void**형이다.

```
const int period = 0;
thread th;
mutex mtx;
bool stopThread = false;
#include <mutex>
```

     - period: 가비지 컬렉터 수행 주기(ms)를 저장하는 변수이다.
     - th: 주기적으로 가비지 컬렉터를 실행하기 위해 쓰레드를 저장하는 변수이다.
     - mtx: 멀티쓰레드 프로그램이기 때문에 instance 와 pointer 의 동기화하기 위해 mutex 를 사용한다.

     stopThread: 쓰레드 종료를 위한 플래그이다.
   - 기본 생성자와 파괴자

```
GarbageCollector() : period(3000) {
    th = thread(&GarbageCollector::timer, this, period);
}
```

     - 초기화 리스트로 period 를 설정한다.
     - 쓰레드를 주기 3000ms(3s)로 실행한다.
     - private 이므로 GetInstance()로 호출된다.

```
~GarbageCollector() {
    stopThread = true;
    if (th.joinable()) {
        th.join();
    }
}
```

- 파괴자가 호출시 쓰레드 종료 플래그를 true 로 변경하고, 쓰레드가 join 이 가능한 경우 join 한다.

- 멤버 함수

```
void timer(int period) {
    while (!stopThread) {
        this_thread::sleep_for(chrono::milliseconds(period));
        collectGarbage();
    }
}
```

- period 만큼의 주기로 가비지 컬렉터를 진행한다.

```
static GarbageCollector& GetInstance() {
    static GarbageCollector instance;
    return instance;
}
```

- 객체를 생성하기 위한 함수

```
// 새로운 포인터 추가
template <typename T>
void addPointer(T** address) {
    lock_guard<mutex> lock(mtx);
    pointer.push_back(reinterpret_cast<void**>(address));
}
```

- 템플릿을 사용하여 여러 타입의 이중포인터를 다룰 수 있게 하였다.
- lock_guard<mutex> lock(mtx): 쓰레드에서 코드를 동기화하기 위해 사용한다.
- 전달받은 이중포인터를 void** 타입으로 명시적 형변환을 하여 포인터 벡터에 추가한다.
- 입력: 임의의 이중포인터, pointer 출력: 임의의 이중포인터가 추가된 pointer

```
// 새로운 객체 추가
void addInstance(void* address) {
    lock_guard<mutex> lock(mtx);
    if (address) {
        instance.push_back(address);
    }
}
```

- 객체의 주소를 받아 객체 벡터의 추가한다.

- nullptr 이 아닌 경우만 추가되도록 한다.
- 입력: 임의의 객체, instance 출력: 임의의 객체가 추가된 instance

```
// 유효하지 않은 포인터 제거
void removePointer() {
    for (auto pptr = pointer.begin(); pptr != pointer.end();) {
        if (*pptr == nullptr) {
            pptr = pointer.erase(pptr);
        } else {
            ++pptr;
        }
    }
}
```

- 벡터에 저장된 포인터 변수가 nullptr 이면 제거한다.
- 입력: pointer 출력: 널포인터가 제거된 pointer

```
// 객체 삭제
void removeInstance(void* p) {
    lock_guard<mutex> lock(mtx);
    for (auto inst = instance.begin(); inst != instance.end();) {
        if (*inst == p) {
            instance.erase(inst);
            break;
        } else {
            ++inst;
        }
    }
}
```

- 임의의 객체를 받아 그 객체가 instance 벡터 존재하면 벡터에서 그 객체를 삭제한다.
- 스마트 포인터의 파괴자에서 호출된다.
- 입력: instance, 임의의 객체 p 출력: 임의의 객체가 제거된 instance

```
// 가비지 컬렉터 실행
void collectGarbage() {
    cout << "가비지 컬렉터 실행" << endl;
    lock_guard<mutex> lock(mtx);
    vector<void*> del_addr;

    removePointer();  // null을 참조하는 포인터 제거

    for (auto inst = instance.begin(); inst != instance.end();) {
        bool isActive = false;  // 활성 객체 여부 확인
        for (auto pptr : pointer) {
            if (*pptr == *inst) {
                isActive = true;
                break;
            }
        }
        if (!isActive) {
            del_addr.push_back(*inst);
            free(*inst);
            inst = instance.erase(inst);

        } else {
            ++inst;
        }
    }
    print_log(del_addr);
}
```

- 비활성 객체를 삭제한다.
- del_addr: 지워진 메모리의 위치를 저장하기 위한 변수이다.
- removePointer(): 컬렉터 실행전 null 을 참조하는 포인터를 제거한다.
- isActive: 활성 객체 여부를 저장한다.
- 이중 for 문

  -> 각 객체에 대해 모든 포인터의 참조 값과 비교하며 활성 객체인지 확인한다.

  -> 활성 객체가 아닌 경우 del_addr 에 추가하고 메모리를 해제한 다음 instance 에서 해당 객체를 삭제한다.
- print_log(del_addr): 제거된 객체 벡터를 전달하여 가비지 컬렉터의 실행 결과를 출력한다.
- 입력: instance, pointer, del_addr, isActive

  출력: 활성객체만 저장된 instance, null 포인터 제거된 poiter, 제거된 객체 추가된 del_addr

```
// 가비지 컬렉터 동작 확인 (delete된 메모리 위치 출력)
void print_log(vector<void*> addr) {
    cout << "Garbage collected." << endl;
    cout << "[ ";
    for (auto i : addr) {
        cout << i << " ";
    }
    cout << "]" << endl;
    cout << "Active objects: " << instance.size() << endl << endl;
}
```

- 가비지 컬레터의 실행 결과 출력
  -> 메모리 해제된 위치, 현재 활성 객체 수
- 입력: 주소 벡터, instance

2. 전역 변수

```
// Global variable 'GC' for the garbage collector class.
GarbageCollector& GC = GarbageCollector::GetInstance();
```

- 전역변수로 가비지 컬렉터 객체 생성

3. 연산자 new 오버로딩

```
// operator new overloading
void* operator new(size_t size) {
    void* ptr = malloc(size);
    return ptr;
}
```

- malloc 으로 사이즈만큼 동적 할당하여 주소 반환

4. smartptr 클래스

- 템플릿으로 선언하여 여러 타입에 대해 사용가능하다.
- 객체 소멸시 자동으로 메모리 해제되는 클래스
- 멤버 변수

  `T* ptr;` - 주소 저장을 위한 변수

- 생성자, 파괴자

```
smartptr(T* p = 0) : ptr(p) { GC.addInstance(ptr); }

~smartptr() {
    GC.removeInstance(ptr);
    delete ptr;
}
```

- 객체 생성 시 자동으로 가비지 컬렉터의 객체 벡터에 추가한다.
- ptr 이 0 인 경우 addInstance 에서 걸러져 추가되지 않는다.

- 객체 소멸 시 자동으로 가비지 컬렉터의 객체 벡터에서 제거되고 메모리 해제도 진행한다.

- 멤버함수

```
T* get_ptr() { return ptr; }
```
: ptr 값 반환한다.

- 대입 연산자 오버로딩

```
smartptr& operator=(T* p) {
    if (ptr) {
        GC.removeInstance(ptr);  // 기존 객체 제거
        delete ptr;              // 기존 메모리 해제
    }
    ptr = p;
    if (ptr) {
        GC.addInstance(ptr);  // 새로운 객체 등록
    }
    return *this;
}
```

  - 기존 객체를 instance 벡터에서 제거 후 기존 메모리를 해제한다.
  - 전달받은 객체를 instance 에 추가한다.

5. main 함수

```
int* ptr;
ptr = new int;
GC.addInstance(ptr);
GC.addPointer(&ptr);
ptr = new int;
GC.addInstance(ptr);
ptr = new int;
GC.addInstance(ptr);
ptr = new int;
GC.addInstance(ptr);
ptr = new int;
GC.addInstance(ptr);
ptr = new int;
GC.addInstance(ptr);
ptr = new int;
GC.addInstance(ptr);
```

```
member* qtr;
qtr = new member;
GC.addInstance(qtr);
GC.addPointer(&qtr);
qtr = new member;
GC.addInstance(qtr);
qtr = new member;
GC.addInstance(qtr);
qtr = new member;
GC.addInstance(qtr);
qtr = new member;
GC.addInstance(qtr);
qtr = new member;
GC.addInstance(qtr);
```

```
// For SmartPjointers
smartptr<member> rtr;
GC.addPointer(reinterpret_cast<void**>(&rtr));
rtr = new member;
rtr = new member;
rtr = new member;
rtr = new member;
rtr = new member;
rtr = new member;

while (1) {
    ptr = new int;
    GC.addInstance(ptr);
    qtr = new member;
    GC.addInstance(qtr);
    rtr = new member;
    Sleep(1000);
}
```

- 세가지 모두 처음 포인터 변수가 생성되면 pointer 벡터에 포인터를 추가해야 한다. 포인터 벡터는 void** 타입이므로 포인터 변수의 주소를 전달해야 한다. 스마트 포인터의 경우 주소의 형변환이 추가로 필요하다.
- ptr 과 qtr 의 경우 객체가 생성될 때마다 instance 벡터에 추가한다.
- 스마트포인터는 대입 연산자 오버로딩을 통해 자동으로 객체가 instance 벡터에 추가된다.
- while 문을 통해 가비지 컬렉터가 주기적으로 잘 실행되는지 확인한다. 오버헤드를 줄이기 위해 1 초씩 sleep 을 걸어준다.

실행 결과

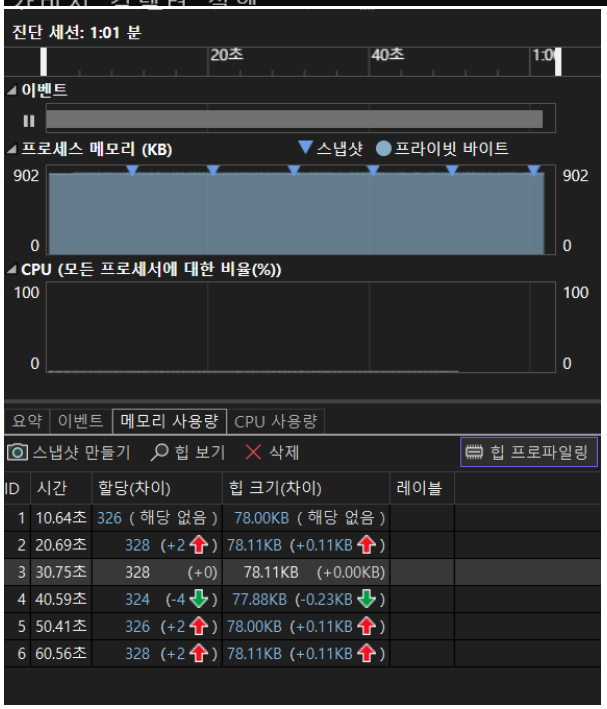
```
가비지 컬렉터 실행
Garbage collected.
[ 000001EAE1A56090 000001EAE1A56210 000001EAE1A561D0 000001EAE1
A56D50 000001EAE1A56D90 000001EAE1A567D0 000001EAE1A565D0 00000
1EAE1A63710 000001EAE1A64390 000001EAE1A63A30 000001EAE1A636C0
000001EAE1A64020 000001EAE1A63530 000001EAE1A63F30 000001EAE1A5
6510 000001EAE1A63760 000001EAE1A56310 000001EAE1A64250 000001E
AE1A563D0 000001EAE1A64340 ]
Active objects: 3

가비지 컬렉터 실행
Garbage collected.
[ 000001EAE1A55F10 000001EAE1A63800 000001EAE1A56310 000001EAE1
A64390 000001EAE1A56D50 000001EAE1A63F80 ]
Active objects: 3

가비지 컬렉터 실행
Garbage collected.
[ 000001EAE1A56A50 000001EAE1A634E0 000001EAE1A56210 000001EAE1
A64390 000001EAE1A55F90 000001EAE1A63580 ]
Active objects: 3

가비지 컬렉터 실행
Garbage collected.
[ 000001EAE1A566D0 000001EAE1A635D0 000001EAE1A56710 000001EAE1
A640C0 000001EAE1A56510 000001EAE1A63B20 ]
Active objects: 3

가비지 컬렉터 실행
```



- 1 분동안 확인해 본 결과 메모리 사용량이 일정량 이상 증가하지 않고, 힙 크기도 거의 일정하게 유지된다.

```
cout << "의도적 실행"<<endl;
GC.collectGarbage();

// For SmartPjointers
smartptr<member> rtr;
GC.addPointer(reinterpret_cast<void**>(&rtr));
rtr = new member;
rtr = new member;
```

```
의도적 실행
가비지 컬렉터 실행
Garbage collected.
[ 000001AF93F96450 000001AF93F96290 000001AF93F96890 000001AF93F96990
 000001AF93F96250 000001AF93F95F10 000001AF93FA44C0 000001AF93FA38E0
000001AF93FA4330 000001AF93FA4510 000001AF93FA41A0 000001AF93FA4470 ]
Active objects: 2

가비지 컬렉터 실행
Garbage collected.
[ 000001AF93F96690 000001AF93FA3D40 000001AF93F96050 000001AF93FA3A70
 000001AF93F96450 000001AF93FA44C0 ]
Active objects: 3

가비지 컬렉터 실행
Garbage collected.
[ 000001AF93F96990 000001AF93FA3B10 000001AF93F96450 000001AF93FA38E0
 000001AF93F95F10 000001AF93FA3890 ]
Active objects: 3

가비지 컬렉터 실행
```

- 스마트포인터가 생성되기 전에 의도적으로 가비지 컬렉터를 호출하여 실행시키는 코드를 추가하여 실행한 것이다.
- 의도적으로 호출한 부분은 아직 포인터 변수가 ptr, qtr 두 개이므로 활성 객체 2 개이고 이후 실행된 가비지 컬렉터는 rtr 도 추가되어 활성 객체가 3 개로 잘 실행된 것을 알 수 있다.

한계 및 개선사항

- 현재 코드는 동적할당될 때마다 직접 instance 에 추가해 주어야한다. 이 경우 코드가 복잡해지면 추가되지 못하고 누락되어 가비지 컬렉터로 관리하지 못할 가능성이 높다.
  ➔ new 연산자 오버로딩으로 이곳에서 instance 에 추가하도록 하면 동적 할당 시마다 자동으로 가비지 컬렉터의 관리를 받을 수 있다.

  - 구현하지 못한 이유: GarbageCollector 클래스에서는 instance 와 pointer, 가비지 컬렉터 실행이 임시 변수 del_addr 이 벡터로 관리되는데 벡터는 데이터를 추가할 때 동적할당을 받기 때문에 전역으로 오버로딩된 new 에 가비지 컬렉터에 rorcpo 를 추가하는 코드를 작성하게 되면 무한 루프에 빠지게 된다. 가비지 컬렉터의 객체를 생성하는 경우또한 동적 할당이 되면서 무한루프에 빠지게 된다. 현재의 지식으로는 아직 무한루프를 해결할 수 있는 방법을 몰라 구현하지 못했지만 이 부분을 해결한다면 더 좋아질 것이다.