객체지향프로그래밍 II

제10장 Class의 이해 및 활용 (Part 4)

1. Dynamic Memory Management with Operators new and delete
2. static Class Members

박인규 교수

1. Dynamic Memory Management with Operators new and delete

동적 메모리 관리 (Dynamic Memory Management)

- 동적 메모리 할당 (allocation) 및 해제 (release)
  - ✓ 내장 데이터 타입이나 사용자 정의 타입에 대해 동적으로 메모리를 할당하고 해제할 수 있는 기능을 제공한다.
  - ✓ new 와 delete 연산자를 통해 이루어진다.
  - ✓ 예를 들어, 고정된 크기의 배열 대신 프로그램 수행 중 크기가 결정되는 (= 동적인) 메모리 공간을 생성
- Heap
  - ✓ 각 프로그램에서 실행 시간에 동적으로 생성되는 객체를 저장하기 위해 할당된 메모리 영역이다.
  - ✓ 반대로, 컴파일 시간에 생성되는 객체는 stack 영역의 메모리를 할당 받음.

동적 메모리 관리 (Dynamic Memory Management)

- 연산자 new
  - ✓ 기본 데이터 타입(int, double 등)이나 클래스 타입의 객체를 실행시간(execution time)에 할당 및 생성
    - 예시
      - Time *ptrTime = new Time;
      - float *ptrNum = new float;
    - 클래스 객체를 생성하는 경우, 동시에 객체의 생성자도 호출
    - 일반적으로 복수의 객체를 생성할 때 흔히 사용함
  - ✓ new의 오른쪽에 지정된 타입의 포인터를 반환
  - ✓ C언어에서의 malloc()과 유사

동적 메모리 관리 (Dynamic Memory Management)

- 연산자 **delete**
  - ✓ 동적으로 할당된 객체를 소멸시킴
    - 이 때, 객체의 소멸자를 호출
    - 예시
      - **delete** ptrTime;
      - **delete** ptrNum;
  - ✓ 객체에 대한 메모리 공간을 해제(release)한다.
    - 해제된 메모리 공간은 다른 객체에 할당되기 위해 재사용이 가능하다.
  - ✓ C언어에서의 free()와 유사

new를 이용한 객체 초기화

- ✓ 새롭게 생성된 기본적인 타입의 객체에 초기값 지정
  - 예시
    - `double *ptr = new double( 3.14159 );`
- ✓ 객체의 생성자에 쉼표로 분리되는 인수 목록을 지정
  - 예시
    - `Time *timePtr = new Time( 12, 45, 0 );`

new 연산자는 동적으로 배열을 할당할 수 있다

- ✓ 10개의 정수 원소를 갖는 배열을 할당

```
int *gradesArray = new int[ 10 ];
```

- ✓ 동적 할당 배열의 크기
  - 프로그램 내에서 정수 변수나 정수 변수의 수식 표현을 통해 나타낼 수 있음

```
int *gradesArray = new int[ count1 + count2 ];
```

동적으로 할당된 배열을 제거

```
delete [] gradesArray;
```

- ✓ 위 문장은 gradesArray가 가리키는 배열에 대한 할당을 해제한다.
- ✓ 위 문장의 포인터가 객체의 배열을 가리키고 있다면,
  - 위 문장은 먼저 배열 내의 모든 객체에 대한 소멸자를 호출한 후, 메모리를 해제한다.
- ✓ 위 문장에서 대괄호([ ])가 없고 gradesArray가 객체의 배열을 가리키고 있다면
  - 배열 내의 첫 번째 객체만 소멸자 호출을 받은 것을 의미한다.
  - 동적 할당받은 배열의 메모리 해제시 [ ]를 누락하지 않도록 조심하시오.

2. static Class Members

static 데이터 멤버

- ✓ 클래스의 모든 객체가 공유하는 변수의 복사본
  - 모든 객체가 같은 변수를 공유 : “Class-wide” 정보
  - 클래스의 특정한 객체의 속성이 아닌 클래스 자체의 속성임
- ✓ 데이터 멤버의 선언시 static 키워드로 시작
- ✓ 비록 전역 변수처럼 보이나, 실제로는 class scope를 가짐
- ✓ public, private or protected 데이터 멤버 모두에 static을 선언될 수 있다.
  - private, protected인 경우 public static member function을 통해 접근

public static 데이터 멤버 (cont)

- ✓ 클래스의 객체가 생성되지 않아도 존재 (클래스 자체의 속성)
  - 클래스의 객체가 존재하지 않을 때 public static 클래스 멤버에 접근하기 위해
    - 클래스의 이름과 이항 스코프 식별 연산자(::)를 사용
    - 예시 - Martian::martianCount
- ✓ 클래스의 어떠한 객체에서도 접근 가능하다.
  - 객체의 이름과 점 연산자(dot operator)(.)를 사용한다.
    - 예시 - myMartian.martianCount

static 멤버 함수

- static 멤버 함수
  - ✓ 클래스의 특정 객체가 아닌 클래스 자체의 서비스
  - ✓ static 이 아닌 클래스 데이터 멤버 또는 멤버 함수에 접근할 수 없다.
  - ✓ static 멤버 함수는 this 포인터를 갖지 않는다.
  - ✓ static 데이터 멤버와 static 멤버 함수는 클래스의 객체와는 독립적으로 존재한다.
    - 그 클래스의 객체도 존재하지 않아도, static 멤버 함수를 호출할 수 있다.

Static Class Members 예제 (Employee.h)

```
1  // Fig. 10.21: Employee.h
2  // Employee class definition.
3  #ifndef EMPLOYEE_H
4  #define EMPLOYEE_H
5
6  class Employee
7  {
8  public:
9     Employee( const char * const, const char * const ); // constructor
10    ~Employee(); // destructor
11    const char *getFirstName() const; // return first name
12    const char *getLastName() const; // return last name
13
14    // static member function
15    static int getCount(); // return number of objects instantiated
16 private:
17    char *firstName;
18    char *lastName;
19
20    // static data
21    static int count; // number of objects instantiated
22 }; // end class Employee
23
24 #endif
```

Function prototype for **`static`** member function

`static` data member keeps track of number of **`Employee`** objects that currently exist

Static Class Members 예제 (Employee.cpp)

```
1  // Fig. 10.22: Employee.cpp
2  // Member-function definitions for class Employee.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <cstring> // strlen and strcpy prototypes
8  using std::strlen;
9  using std::strcpy;
10
11 #include "Employee.h" // Employee class definition
12
13 // define and initialize static data member at file scope
14 int Employee::count = 0;
15
16 // define static member functic
17 // Employee objects instantiate
18 int Employee::getCount()
19 {
20    return count;
21 } // end static function getCount
```

`static` data member is defined and initialized at file scope in the **`.cpp`** file

`static` member function can access only **`static`** data, because the function might be called when no objects exist

Static Class Members 예제 (Employee.cpp)

```
22
23 // constructor dynamically allocates space for first and last name and
24 // uses strcpy to copy first and last names into the object
25 Employee::Employee( const char * const first, const char * const last )
26 {
27    firstName = new char[ strlen( first ) + 1 ];
28    strcpy( firstName, first );
29
30    lastName = new char[ strlen( last ) + 1 ];
31    strcpy( lastName, last );
32
33    count++; // increment static count of employees
34
35    cout << "Employee constructor for " << firstName
36       << ' ' << lastName << " called." << endl;
37 } // end Employee constructor
38
39 // destructor deallocates dynamically allocated memory
40 Employee::~Employee()
41 {
42    cout << "~Employee() called for " << firstName
43       << ' ' << lastName << endl;
44
45    delete [] firstName; // release memory
46    delete [] lastName; // release memory
47
48    count--; // decrement static count of employees
49 } // end ~Employee destructor
```

Dynamically allocating **`char`** arrays

Non-**`static`** member function (i.e., constructor) can modify the class's **`static`** data members

Deallocating memory reserved for arrays

Static Class Members 예제 (Employee.cpp)

```
50
51 // return first name of employee
52 const char *Employee::getFirstName() const
53 {
54    // const before return type prevents client from modifying
55    // private data; client should copy returned string before
56    // destructor deletes storage to prevent undefined pointer
57    return firstName;
58 } // end function getFirstName
59
60 // return last name of employee
61 const char *Employee::getLastName() const
62 {
63    // const before return type prevents client from modifying
64    // private data; client should copy returned string before
65    // destructor deletes storage to prevent undefined pointer
66    return lastName;
67 } // end function getLastName
```

Static Class Members 예제 (driver)

```
1  // Fig. 10.23: fig10_23.cpp
2  // Driver to test class Employee.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include "Employee.h" // Employee class definition
8
9  int main()
10 {
11    // use class name and binary scope resolution operator to
12    // access static number function getCount
13    cout << "Number of employees before instantiation of any objects is "
14       << Employee::getCount() << endl; // use class name
15
16    // use new to dynamically create two new Employ
17    // operator new also calls the object's construc
18    Employee *e1Ptr = new Employee( "Susan", "Baker" );
19    Employee *e2Ptr = new Employee( "Robert", "Jones" );
20
21    // call getCount on first Employee object
22    cout << "Number of employees after objects
23       << e1Ptr->getCount();
24
25    cout << "\n\nEmployee 1: "
26       << e1Ptr->getFirstName() << " " <<
27       << "\nEmployee 2: "
28       << e2Ptr->getFirstName() << " " << e2Ptr->getLastName() << "\n\n";
```

Calling **`static`** member function using class name and binary scope resolution operator

Dynamically creating **`Employee`**s with **`new`**

Calling a **`static`** member function through a pointer to an object of the class

Static Class Members 예제 (driver)

```
29
30    delete e1Ptr; // deallocate memory
31    e1Ptr = 0; // disconnect pointer from free-store space
32    delete e2Ptr; // deallocate memory
33    e2Ptr = 0; // disconnect pointer fro
34
35    // no objects exist, so ca
36    // using the class name and the binary scope resolution operator
37    cout << "Number of employees after objects are deleted is "
38       << Employee::getCount() << endl;
39    return 0;
40 } // end main
```

Releasing memory to which a pointer points

Disconnecting a pointer from any space in memory

```
Number of employees before instantiation of any objects is 0
Employee constructor for Susan Baker called.
Employee constructor for Robert Jones called.
Number of employees after objects are instantiated is 2

Employee 1: Susan Baker
Employee 2: Robert Jones

~Employee() called for Susan Baker
~Employee() called for Robert Jones
Number of employees after objects are deleted is 0
```

7주차 1교시 수업을 마칩니다.