Lecture #06

Class (2)

SE271 Object-Oriented Programming (2020)

Yeseong Kim

Original slides from Prof. Shin at DGIST

Short Notice

- HW1 due is tonight! ☺
- HW2 will be released today! ☹
- The instruction for the project proposal will be released on Wednesday
  - Find your teammate
    (during this crisis ☹, but we should proceed ☺)
  - Four members in a team
  - Utilize Piazza used in tutoring
- You can get a bonus point for HW1 until today!

Today's Topic

- Overloading
- Accessor again
- Constructor and Destructor
- Special Pointer: this
- Initialization
- Const function

Function Overloading

- The same task with different parameters
  - sum : 1 + 3 — `int Sum ( int, int);`
  - sum : 1 + 3.5 — `double Sum ( int, double);`
  - sum : 1.5 + 3 — `double Sum (double , int);`
  - sum : 1 + 3 + 2 — `int Sum ( int, int, int);`
  - sum : ‘a’ + ‘b’ — `char* Sum (char, char);`

- **Function Overloading**
  - the same named Functions with different parameters (type/number)

What about return type?

Example: sum

```
int Sum ( int, int);
double Sum ( int, double);
double Sum (double , int);
int Sum ( int, int, int);
char* Sum (char, char);
```

```
int Sum(int n1, int n2) { return n1 + n2; }
double Sum(int n1, double n2)
{ return n1 + n2; }
double Sum(double n1, int n2)
{ return n1 + n2; }
int Sum(int n1, int n2, int n3)
{ return n1 + n2+n3; }

char * Sum(char c1, char c2) {
    char* ret = new char[3]{ c1, c2, '\0' };
    return ret;
}
```

Ambiguity : with default parameters

```
int Sum ( int, int);
int Sum ( int, int, int=0);

int Sum(int n1, int n2){
    return n1 + n2;
}
int Sum(int n1, int n2, int n3) {
    return n1 + n2 + n3;
}
```

```
int main() {
    cout << Sum(1, 2, 3) << endl;
    cout << Sum(1, 2) << endl;
     return 0;
}
```

Implicit type conversion?

Ambiguity : Implicit type conversion

```
int Sum ( int, int);
float Sum ( int, float);

int Sum (int n1, int n2){
    return n1+n2;
}
float Sum (int n1, float n2) {
    return n1+n2;
}
```

```
int main() {
    cout << Sum(1, 2) << endl;
    cout << Sum(1, 2.0f) << endl;

    cout << Sum(1, 2.0) << endl;
      return 0;
}
```

[AGAIN] C vs. C++: Accessor

- **Syntax**
  - Usage (cf. struct)

```
struct  Student {
    int id;
    string name;
};
Student students[10];
Student * student1;
student1 = new Student{201911999, “Jo

cout << students[0].id << endl;
cout << (*student1).id << endl;
cout << student1->id << endl;

delete student1;
```

Problematic Code

```
class  Student {
private:
    int id;
    string name;
public:
    int GetID() { return id; }
};
Student students[10];
Student* student1;
student1 = new Student{ 201911999, "John" };

cout << students[0].id << endl;
cout << (*student1).id << endl;
cout << student1->id << endl;
delete student1;
```

Class: Constructor and Destructor

```
class  Student {
private:
    int m_id;
    string m_name;
public:
    int GetID() { return m_id; }
    void SetID(int ID) {m_id = ID;}
};

Student s1;
Student* s2;
s2 = new Student;
delete s2;
```

- Instance creation
  - Memory allocation
  - Call the constructor
- Instance deletion
  - Call the destructor
  - Memory deallocation

Class: Constructor

- Constructor
  - Initialization for the instance
    - Member variables, Memory allocation, file open, network connection
  - The same name as its class
  - No return value
  - Probably More than one constructor (a **default constructor** may be created by compiler)

```
class  Student {
private:
    int* m_pID;
    std::string m_name;
public:
    Student();
    Student(int, string);
};
```

```
Student::Student() {
    m_pID = new int(0);
    m_name = "Alice";
}
Student::Student(int id, std::string name) {
    m_pID = new int(id);
    m_name = name;
}
```

Class: Destructor

- Destructor
  - When the instance is deleted
    - Memory deallocation, file close, network disconnection
  - '~' + The same name as its class
  - No return value
  - Only one destructor (a **default destructor** may be created by compiler)

```
class  Student {
private:
    int* m_pId;
    std::string m_name;
public:
    Student();
    ~Student();
};
```

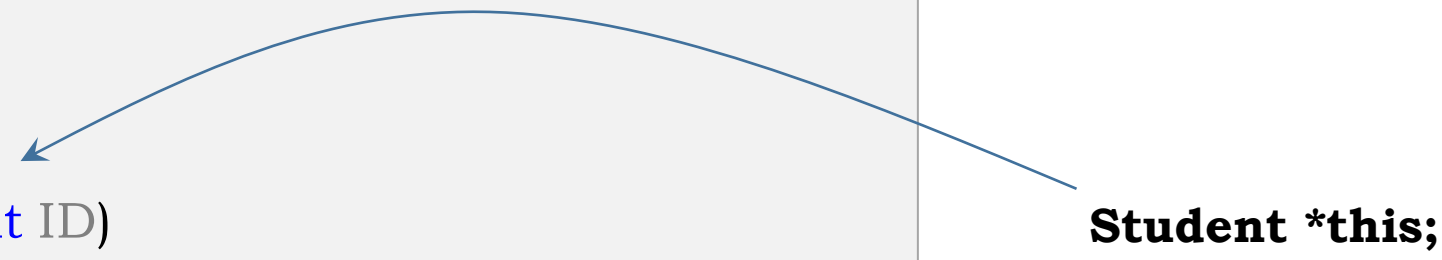
```
Student::Student() {
    m_pId = new int(0);
    m_name = "Alice";
}
Student::~Student() {
    delete m_pId;
}
```

Class: this pointer

- **'this' pointer points at itself**

```
class Student {
private:
    int m_id;
    char m_name;
public:
    int GetID() { return m_id; }
    void SetID(int);
};

void Student::SetID(int ID)
{
    this->m_id = ID;        // m_id = ID; (*this). m_id = ID;
}
```



Class: Member Variable Initialization(1)

1. Class method body

```
class  Student {
private:
    int *m_pId;
    std::string m_name;
public:
    void SetVariables(int, string);
};

void Student::SetVariables(int id, string name){
    this->m_pId = new int(id);
    m_name = name;
}
```

```
int main() {
    Student s1;
    s1.SetVariables(201911999, “Alice”);
    Student s2;
    s2.SetVariables(201911998, “Bob”);

    return 0;
}
```

Class: Member Variable Initialization(2)

2-1. By Constructor w/ or w/o initializer list

```
class  Student {
private:
    int * m_pId;
    std::string m_name;
public:
//    void SetVariables(int, string);
    Student( );
    ~Student();
};
Student::Student (){
    this->m_pId = nullptr;   // = new int{0};
    m_name = “”;
}
```

```
Student::Student () : m_pId{nullptr}, m_name{“”}
{
}

Student::~Student ()
{
    if (! m_pId ) {
        delete m_pId;
        m_pId = nullptr;
    }
}

Student s1;
```

Class: Member Variable Initialization(2)

2-2. By **Other Constructors** w/ or w/o initializer list

```
class  Student {
private:
    int * m_pId;
    std::string m_name;
public:
    Student( );
    Student(int, string );
    ~Student();
};
Student::Student () : m_pId{nullptr}, m_name{“”} { }
Student::~Student () {
    if (! m_pId ) { delete m_pId; m_pId = nullptr; }
}
```

```
Student::Student (int id, string name)
{
    m_pId = new int{id};
    m_name = name;
}

Student::Student (int id, string name) :
m_pId{new int(id) }, m_name{name}
{
}

Student::Student () : Student(0, “”) { }
```

Class: Member Variable Initialization(3)

3. Since C++11

```
class  Student {
private:
    int * m_pId  = nullptr ;
    std::string m_name = “” ;
public:
    Student( );
    ~Student();
};
Student::Student () { }
Student::~Student () {
    if (! m_pId ) { delete m_pId; m_pId = nullptr; }
}
```

References

- Learn c++
  - https://www.learncpp.com/
  - Chapter 8

ANY QUESTIONS?