Lecture 8: Abstraction II:
Special Member Functions

Introduction

Special member functions are class methods defined by the compiler during compilation if not explicitly defined that help manage resources and ensure objects are properly initialized, copied, and cleaned up.

Constructors

Constructors are methods that initialize the instance fields of a class and perform any required setup. They have no return type, return nothing, and are called when an object is instantiated (created). Their general syntax is

`[explicit]` *`class`*`(`*`parameter-list`*`)[:`*`member-initialization-list`*`]{`*`body`*`}`

Object Declaration When an object is instantiated, it can invoke a constructor using any one of the following syntaxes:

1. *`class identifier`*`(`*`argument-list`*`);`
   - parentheses must be omitted to invoke the default constructor.
2. *`class identifier`*` = {`*`argument-list`*`};`
   - cannot be used to invoke the default constructor.
   - curly braces can be omitted to invoke a constructor with a single parameter.
3. *`class identifier`*` = `*`class`*`(`*`argument-list`*`);`

where the second format is prohibited if the constructor is defined with the explicit keyword.

Member Initialization List The member initialization list either allows the initialization of instance fields or invocation of another constructor, a process called *constructor delegation*, before executing the constructor's body. When used to initialize fields, the field initialization syntax is

- **Variable syntax**: *`field`*`(`*`argument-list`*`)`

  where *argument-list* are arguments for a constructor of *field*.
- **Array syntax**: *`field`*`{`*`argument-list`*`}`

Furthermore, when used for constructor delegation, it can invoke either another constructor of the class or a constructor of one of its base classes if any [elaboration of base classes is in the inheritance lecture] with the syntax is

`class``(`*`argument-list`*`)`

which is an *anonymous object*.

Default Constructor The *default constructor* is the primary constructor and a special member function. It is invoked whenever a standard variable declaration is used. Its syntax is

`[explicit]` *`class`*`()[:`*`member-initialization-list`*`]{`*`body`*`}`

All other constructors are considered *overloaded constructors*.

Copy Constructor Another special member function constructor is called the *copy constructor*. It creates a new object as a copy of an existing object. Its syntax is

[explicit]*class*(const *class*& *identifier*)[: *member-initialization-list*]{*body*}

The copying procedure can either be a *shallow copy* [*memberwise copy*] or a *deep copy*. A shallow copy copies field content from the parameter to the object, which may lead to shared resources among objects when dealing with pointers since the content of a pointer is an address.

Whereas, a deep copy copies resources, which means its pointer fields allocate new memory and then copy the dereferenced content.

Example:
Two versions using different copy procedures including all possible object instantiation formats.

```
class SV
{
  public:
  int *x;
  char id;
  SV() : SV(0) {}
  explicit SV(int y) : (new int(y)), id('x') {}
  SV(int x,char id) : SV(x) {this->id = id;}
  SV(const SV& obj) : id(obj.id)
  {
    //shallow copy
    x = obj.x;
  }
};

int main()
{
  //default constructor
  SV a1, a3 = SV();
  //1st overloaded constructor
  SV b1(4), b3 = SV(4);
  //2nd overloaded constructor
  SV c1(8,'t'), c2 = {8,'t'}, c3 = SV(8,'t'),;
  //copy constructor
  SV d1(c1), d2 = c2, d3 = SV(c3);
  return 0;
}
```

```
class DV
{
  public:
  int *x;
  char id;
  DV() : DV(0) {}
  explicit DV(int y) : (new int(y)), id('x') {}
  DV(int x,char id) : DV(x) {this->id = id;}
  DV(const DV& obj) : id(obj.id), x(new int)
  {
    //deep copy
    *x = *(obj.x);
  }
};

int main()
{
  //default constructor
  DV a1, a3 = DV();
  //1st overloaded constructor
  DV b1(4), b3 = DV(4);
  //2nd overloaded constructor
  DV c1(8,'t'), c2 = {8,'t'}, c3 = DV(8,'t'),;
  //copy constructor
  DV d1(c1), d2 = c2, d3 = DV(c3);
  return 0;
}
```

Destructor

The *destructor* is a special member function used for resource cleanup (*garbage collection*). It deals with memory deallocation, stream closures, and other termination operations. Its syntax is

~*class*(){*body*}

It is invoked whenever the object's scope ends or when the object is explicitly deallocated if it was explicitly allocated.

Example:
Destructors of the classes from the previous example.

```
class SV
{
  //previous code
  public:
  ~SV() {delete x;}
};
```

```
class DV
{
  //previous code
  public:
  ~DV() {delete x;}
};
```

Using the current destructor, *SV* objects that use the copy constructor may have issues with field *x* since it performs a shallow copy.

Assignment Operator

The *assignment operator* is the last of the special member function. It copies the content of another object to the object. Its syntax is

```
class& operator=(const class& identifier){body}
```

It is invoked when its object is assigned another class object after being instantiated; hence, its definition normally confirms that the argument and the object are different before performing the copy as illustrated in the template code below

```
class& operator=(const class& identifier)
{
  if(this != &identifier)
  {
    body
  }
  return *this;
}
```

Anyway, the copy procedure is identical to the copy constructor, except sometimes with pointers; it needs to deallocate memory before allocating new memory to prevent memory leaks. Furthermore, it always returns the dereferenced this pointer.

Example:
Assignment operator of the classes from the previous examples.

```
class SV
{
  public:
  //previous code
  SV& operator=(const SV& rhs)
  {
    if(this != &rhs)
    {
      //shallow copy
      delete x;
      x = rhs.x;
      id = rhs.id;
    }
    return *this;
  }
};


int main()
{
  //previous code
  SV *s = new SV(6);
  a1 = SV(81); //assignment operator call
  delete s; //destructor call
  return 0;
}
```

```
class DV
{
  public:
  //previous code
  DV& operator=(const DV& rhs)
  {
    if(this != &rhs)
    {
      //deep copy
      delete x;
      x = new int;
      *x = *(rhs.x);
      id = rhs.id;
    }
    return *this;
  }
};


int main()
{
  //previous code
  DV *s = new DV(6);
  a1 = DV(81); //assignment operator call
  delete s; //destructor call
  return 0;
}
```