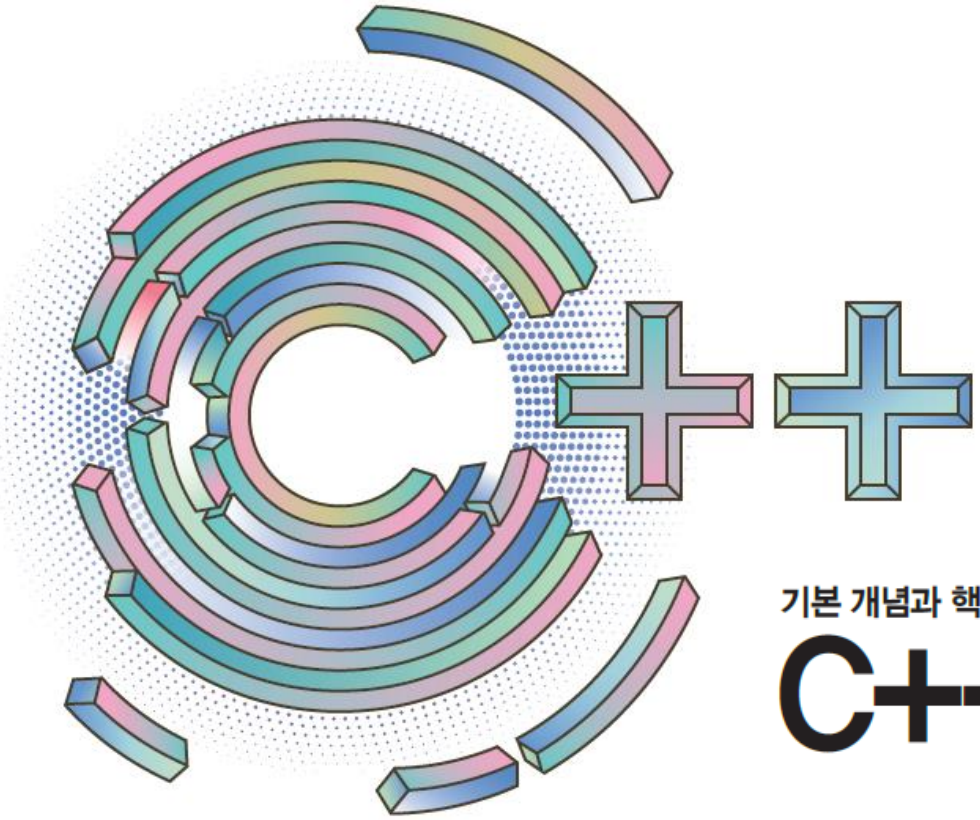
한빛아카데미 Hanbit Academy, Inc.



기본 개념과 핵심 원리로 배우는

C++ 프로그래밍

12장. 상속

목차

01. 자식(파생) 클래스

■ 상속성의 개념

- C++ 클래스의 상속이 가능해지면서 비로소 코드의 재활용이 가능해진다.
- 클래스에 있어서 상속은 추상적인 것을 좀 더 구체적인 것으로 만드는 과정을 나타내기도 한다.

01. 자식(파생) 클래스

■ 자식(파생) 클래스의 개념

```
class 클래스명 : 접근지정자1 부모클래스1 [, 접근지정자2 부모클래스2 ...]
{
   // 클래스 정의
};
```

```
class CTest : public CParent1, public CParent2
{
   ...
}
```

01. 자식(파생) 클래스

■ 멤버의 상속

[예제 12-1] 멤버의 상속

```
1  #include <iostream>
2  using namespace std;
3
4  class CParent
5  {
6  public:
7     void Func1()
8     {
9        m_Value1 = 1;
10    }
11
12    int m_Value1;
13 };
```

01. 자식(파생) 클래스

■ 멤버의 상속

```
14
15  class CTest : public CParent
16  {
17  public:
18      void Func1()   // 재정의
19      {
20          m_Value1 = 2;
21      }
22
23      void Func2()   // 새로 정의
24      {
25          m_Value2 = 3;
26      }
27
28      int m_Value1;    // 재정의
```

01. 자식(파생) 클래스

■ 멤버의 상속

```
29      int m_Value2;    // 새로 정의
30  };
31
32  void main()
33  {
34      CTest t;
35      t.CParent::Func1();
36      t.Func1();
37      t.Func2();
38
39      cout << t.CParent::m_Value1 << endl;
40      cout << t.m_Value1 << endl;
41      cout << t.m_Value2 << endl;
42  }
```

```
1
2
3
```

02. 접근 지정자

■ 접근 지정자의 역할

- 클래스는 외부 함수에 자신의 멤버에 대한 공개 여부를 결정할 수 있어야 하며 그런 역할을 하는 것이 바로 접근 지정자이다.

- 자기자신 클래스에서 정의된 멤버에 사용되는 것을 **'멤버 접근 지정자'**라고 하며, 부모 클래스로부터 상속받은 멤버에 사용되는 것을 **'상속 접근 지정자'**라고 한다.

02. 접근 지정자

■ 멤버 접근 지정자

[예제 12-2] 멤버 접근 지정자

```
1  class CTest
2  {
3  public:
4     void Func()
5     {
6        m_Public++;    // OK
7        m_Protected++;    // OK
8        m_Private++;    // OK
9     }
10
11    int m_Public = 1;    // Public
12
13 protected:
```

02. 접근 지정자

■ 멤버 접근 지정자

```
14      int m_Protected = 2;    // Protected
15
16  private:
17      int m_Private = 3;    // Private
18  };
19
20  class CChild : public CTest
21  {
22  public:
23      void Func()
24      {
25          m_Public++;    // OK
26          m_Protected++;    // OK
27          m_Private++;    // Error
28      }
```

02. 접근 지정자

■ 멤버 접근 지정자

```
29  };
30
31  void main()
32  {
33      CTest t;
34      t.m_Public++;    // OK
35      t.m_Protected++;    // Error
36      t.m_Private++;    // Error
37  }
```

02. 접근 지정자

■ 멤버 접근 지정자

[예제 12-3] protected 멤버의 접근

```
1  class CTest
2  {
3  protected:
4     int m_Value;
5  };
6
7  class CChild : public CTest
8  {
9  public:
10    void Set(int arg)
11    {
12       m_Value = arg;
13    }
```

02. 접근 지정자

■ 멤버 접근 지정자

```
14  };
15
16  void main()
17  {
18      CTest t;
19      t.m_Value = 1;    // Error
20
21      CChild& c = (CChild&)t;
22      c.Set(1);  // OK
23  }
```

02. 접근 지정자

■ 상속 접근 지정자

구분		자식(파생) 클래스 – 상속 접근 지정자		
		public 상속	private 상속	protected 상속
부모 클래스 – 멤버 접근 지정자	public	public	private	protected
	private	접근 불가	접근 불가	접근 불가
	protected	protected	private	protected

[상속 멤버의 접근 지정자]

02. 접근 지정자

■ 상속 접근 지정자

[예제 12-4] 상속 접근 지정자 protected - 상속 멤버의 접근

```
1  class CTest
2  {
3  public:
4     void Func()
5     {
6        m_Public++;    // OK
7        m_Protected++;    // OK
8        m_Private++;    // OK
9     }
10
11    int m_Public = 1;    // Public
12
13 protected:
```

02. 접근 지정자

■ 상속 접근 지정자

```
14      int m_Protected = 2;    // Protected
15
16  private:
17      int m_Private = 3;    // Private
18  };
19
20  class CChild : protected CTest
21  {
22  public:
23      void Func()
24      {
25          m_Public++;    // OK
26          m_Protected++;    // OK
27          m_Private++;    // Error
28      }
```

02. 접근 지정자

■ 상속 접근 지정자

```
29  };
30
31  void main()
32  {
33      CChild c;
34      c.m_Public++;     // Error
35      c.m_Protected++;   // Error
36      c.m_Private++;    // Error
37  }
```

02. 접근 지정자

friend

[예제 12-5] 상속 접근 지정자 protected - 상속 멤버의 접근

```
1  #include <iostream>
2  using namespace std;
3
4  class CTest
5  {
6     friend void main();
7     friend class COther;
8
9  private:
10    int m_Value;
11 };
12
13 class COther
```

02. 접근 지정자

friend

```
14  {
15  public:
16      void Func(CTest& obj)
17      {
18          obj.m_Value++;
19      }
20  };
21
22  void main()
23  {
24      CTest t;
25      t.m_Value = 1;
26
27      COther o;
```

02. 접근 지정자

friend

```
28      o.Func(t);
29
30      cout << t.m_Value << endl;
31  }
```

02. 접근 지정자

friend

[예제 12-6] friend – 전역 연산자 함수

```
1  class CTest
2  {
3     friend CTest operator + (int arg1, const CTest& arg2);
4
5  public:
6     CTest(int arg)
7     {
8        m_Value = arg;
9     }
10
11 private:
12    int m_Value = 0;
13 };
```

02. 접근 지정자

friend

```
14
15  // 전역 연산자 함수만 가능
16  CTest operator + (int arg1, int CTest& arg2)
17  {
18     CTest t(0);
19     t.m_Value = arg1 + arg2.m_Value;   // ?
20     return t;
21  }
22
23  void main()
24  {
25     CTest t(2);
26     CTest s = 1 + t;
27  }
```

03. 상속되지 않는 함수

■ 생성자와 소멸자

[예제 12-7] 생성자 상속

```
1  class CParent
2  {
3  public:
4     CParent(int arg)
5     {
6     }
7  };
8
9  class CTest : public CParent
10 {
11 };
12
13 void main()
```

03. 상속되지 않는 함수

■ 생성자와 소멸자

```
14  {
15      CParent p(0);   // OK
16      CTest t(0);   // Error
17  }
```

03. 상속되지 않는 함수

■ 대입 연산자 함수

[예제 12-8] 연산자 함수 상속

```
1  class CParent
2  {
3  public:
4     CParent& operator = (int arg)
5     {
6        m_Value = arg;
7        return *this;
8     }
9
10    CParent& operator ++ ()
11    {
12       m_Value++;
13       return *this;
```

03. 상속되지 않는 함수

■ 대입 연산자 함수

```
14      }
15
16      CParent operator + (int arg)
17      {
18          CParent p = *this;
19          p.m_Value += arg;
20          return p;
21      }
22
23      int m_Value = 0;
24  };
25
26  class CTest : public CParent
27  {
```

03. 상속되지 않는 함수

■ 대입 연산자 함수

```
28  };
29
30  void main()
31  {
32      CParent p;
33      p = 1;    // OK
34      ++p;    // OK
35      p + 1;    // OK
36
37      CTest t;
38      t = 1;    // Error
39      ++t;    // OK
40      t + 1;    // OK
41  }
```

04. 다중 상속

■ 다중 상속의 문제점

[예제 12-9] 다중 상속 다이아몬드 구조

```
1  class CTop
2  {
3  public:
4     int m_Top;
5  };
6
7  class CLeft : public CTop
8  {
9  public:
10    int m_Left;
11 };
12
13 class CRight : public CTop
```

04. 다중 상속

■ 다중 상속의 문제점

```
14  {
15  public:
16     int m_Right;
17  };
18
19  class CBottom : public CLeft, public CRight
20  {
21  public:
22     CBottom()
23     {
24        m_Top = 1;   // Error
25        m_Left = 2;
26        m_Right = 3;
27        m_Bottom = 4;
28     }
```

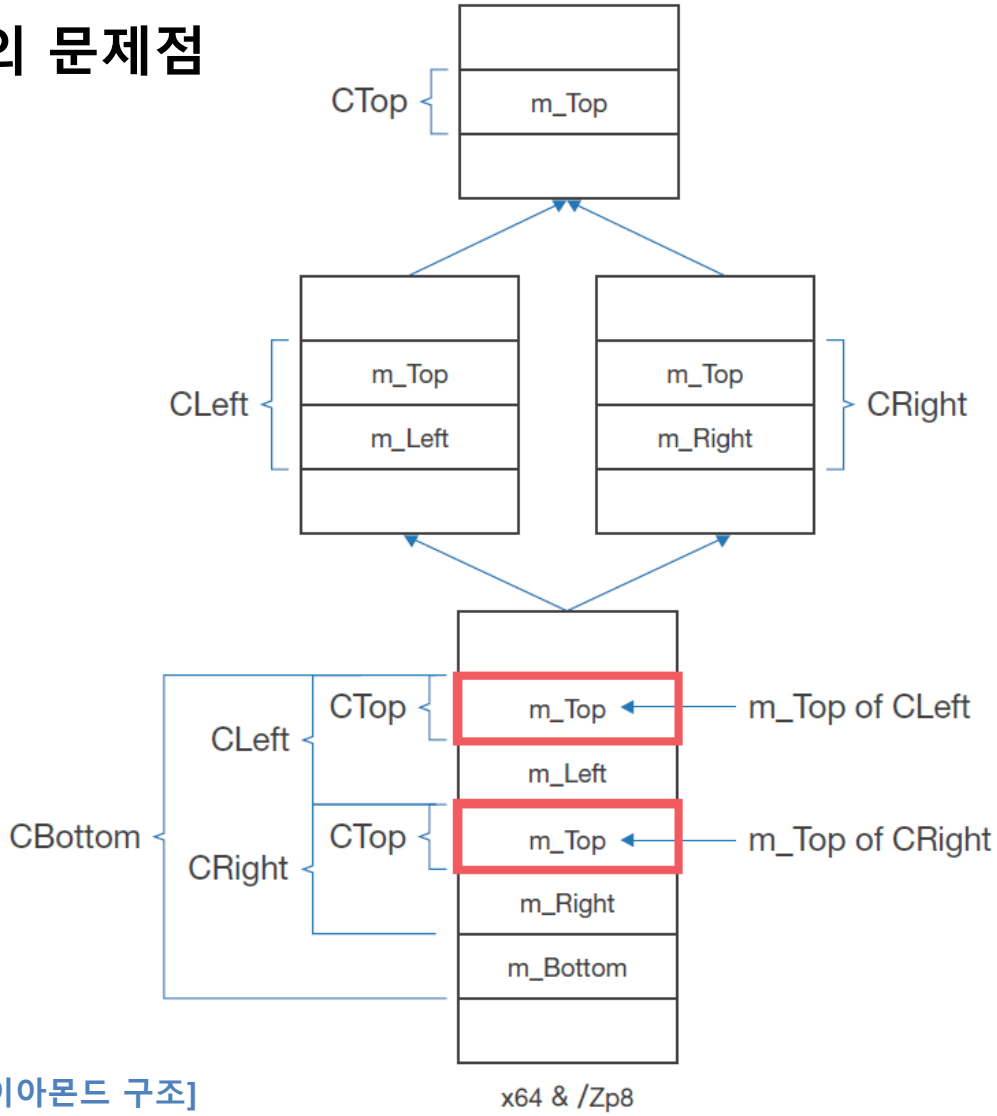
04. 다중 상속

■ 다중 상속의 문제점

```
29
30      int m_Bottom;
31  };
32
33  void main()
34  {
35      CBottom b;
36  }
```

04. 다중 상속

■ 다중 상속의 문제점



[다중 상속 – 다이아몬드 구조]

05. 가상 상속

■ 가상 기저 클래스

[예제 12-10] 가상 기저 클래스

```
1  class CTop
2  {
3  public:
4     int m_Top;
5  };
6
7  class CLeft : virtual public CTop
8  {
9  public:
10    int m_Left;
11 };
12
13 class CRight : virtual public CTop
```

05. 가상 상속

■ 가상 기저 클래스

```
14  {
15  public:
16     int m_Right;
17  };
18
19  class CBottom : public CLeft, public CRight
20  {
21  public:
22     CBottom()
23     {
24        m_Top = 1;    // OK
25        m_Left = 2;
26        m_Right = 3;
27        m_Bottom = 4;
28     }
```

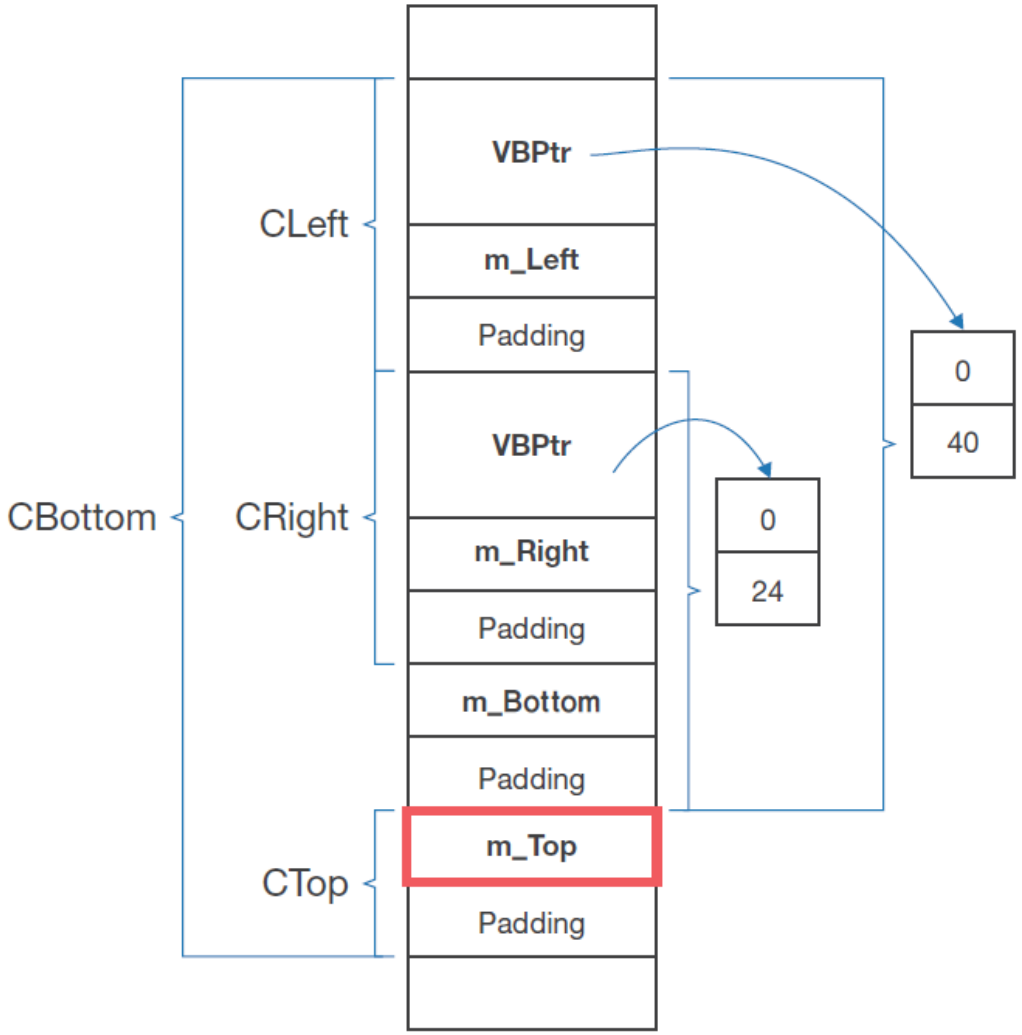
05. 가상 상속

■ 가상 기저 클래스

```
29
30      int m_Bottom;
31  };
32
33  void main()
34  {
35      CBottom b;
36  }
```

05. 가상 상속

■ 가상 기저 클래스



[가상 기저 클래스] x64 & /Zp8

05. 가상 상속

■ 가상 기저 클래스의 생성자

[예제 12-11] 가상 기저 클래스의 생성자

```
1  #include <iostream>
2  using namespace std;
3
4  class CTop
5  {
6  public:
7     CTop()
8     {
9        cout << "CTop" << endl;
10    }
11 };
12
13 class CLeft : virtual public CTop
```

05. 가상 상속

■ 가상 기저 클래스의 생성자

```
14  {
15  public:
16     CLeft()
17     {
18        cout << "CLeft" << endl;
19     }
20  };
21
22  class CRight : virtual public CTop
23  {
24  public:
25     CRight()
26     {
27        cout << "CRight" << endl;
28     }
29  };
```

05. 가상 상속

■ 가상 기저 클래스의 생성자

```
30
31  class CBottom : public CLeft, public CRight
32  {
33  public:
34      CBottom()
35      {
36          cout << "CBottom" << endl;
37      }
38  };
39
40  void main()
41  {
42      CBottom b;
43  }
```

```
CTop
CLeft
CRight
CBottom
```

Thank You !

한빛아카데미 Hanbit Academy, Inc.