객체지향설계
Fall 2018

임성수 교수

Lecture 4: Making and Using Objects

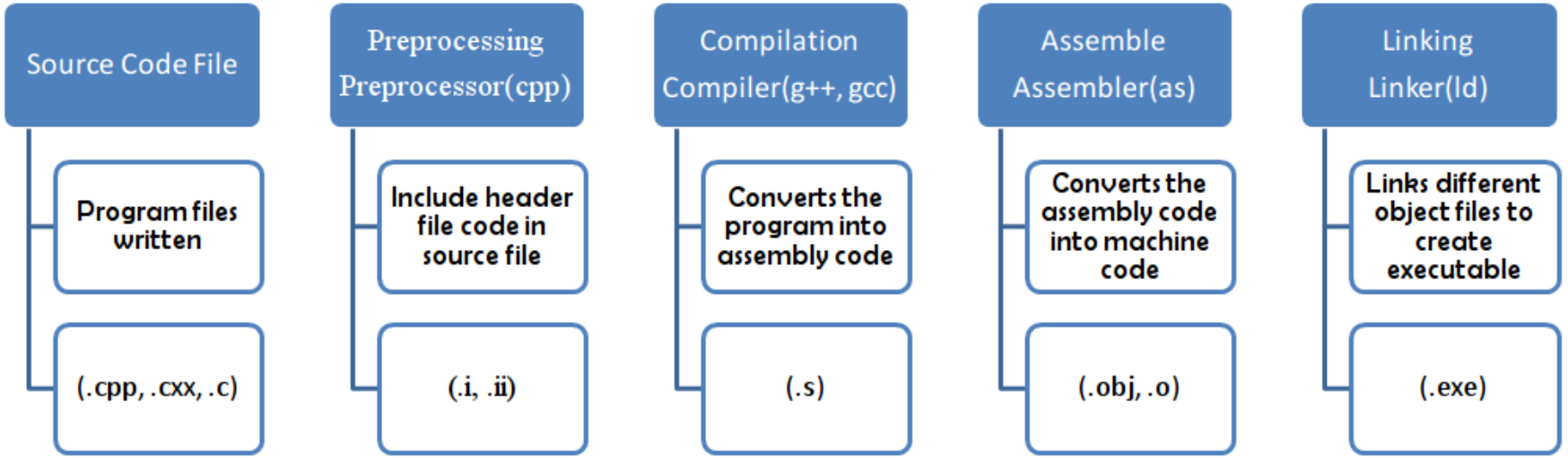
수업 내용

1. 언어 변환 절차
2. 분리 컴파일
3. C++ 프로그램 작성
4. iostream 클래스
5. 문자열
6. 파일 읽기 & 쓰기
7. 벡터

1. 언어 변환 절차

C/C++ 프로그램 실행 절차

- 소스코드 → 전처리기 → **컴파일러** → 어셈블러 → 링커 → 실행 파일
  - 전처리기: 앞서 선언된 지시자들(directives) 처리
  - 컴파일러: 고급 언어를 어셈블리어로 번역
  - 어셈블러: 어셈블리어를 기계어로 번역



1. 언어 변환 절차

전처리 (Preprocessing)

- **전처리기**: 컴파일 직전에 실행되는 별도의 프로그램
- 같은 작업을 반복해야 할 때 헤더 파일 삽입
- 사용자 정의 헤더 파일 사용 가능

전처리 지시자

- 파일 포함(#include)
  - #include 〈파일명〉 // 지정된 디렉토리에서 검색 후 프로그램에 추가
  - #include "파일명" // 해당 파일을 찾아서 프로그램에 추가
- 매크로 정의 (#define)
  - #define 상수명 상수값 // [예제] #define PI 3.14
  - #define 함수명 함수 // [예제] #define MULT(a,b) ((a)*(b))

1. 언어 변환 절차

인터프리터 (Interpreter)

- 소스코드를 단계별로 실행해주는 프로그램
- **명령 한 줄씩 번역**하는 방식으로 번역 진행
- 프로그램 수정이 간편하고 디버깅에 유리

[예제] Python, JavaScript

컴파일러 (Compiler)

- 소스코드를 저급 언어로 **한꺼번에 번역**해주는 프로그램
- 어셈블리어 형태의 목적 모듈(object module) 생성
- 분리 컴파일: 소스코드를 여러 파일에 저장 후 분리하여 컴파일

[예제] C/C++, Java

1. 언어 변환 절차

컴파일러와 인터프리터의 차이점

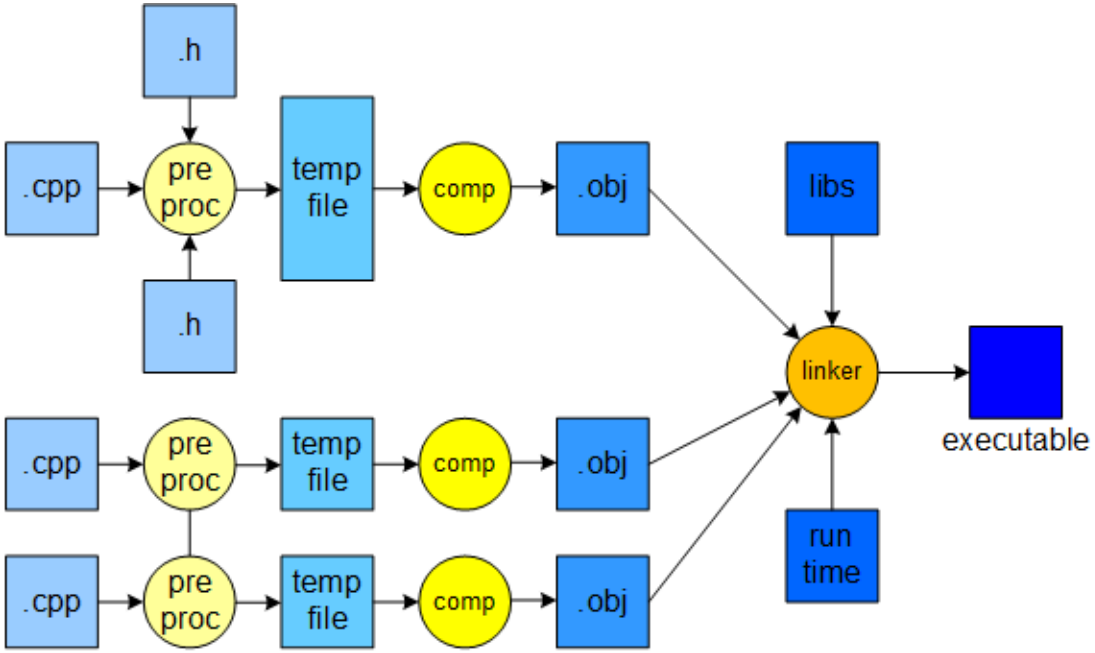
- 인터프리터: **수정이 빈번한 경우** 유리
- 컴파일러: **실행 속도** 우위
- 목적 모듈을 만드는 이유
  - 작업 분업화, 프로그램 모듈화, 확장성에 유리
  - 소스코드의 보안을 유지하며 기능 제공

	인터프리터	컴파일러
번역 단위	한 줄씩	전체
실행 속도	느림	빠름
목적 모듈	생성하지 않음	생성함
메모리 할당	할당하지 않음	목적 모듈 생성 시

1. 언어 변환 절차

분리 컴파일과 링커

- **분리 컴파일** (Separate-): 각 부분 프로그램을 개별적으로 컴파일
- **링커**: 목적 모듈, 라이브러리 등을 결합하여 실행 파일 생성



1. 언어 변환 절차

링커와 라이브러리

- **링커**: 목적 모듈(.obj)을 연결해 실행 파일(.exe)을 만드는 프로그램
- **라이브러리 사용**
  1. #include 〈파일명〉 // 라이브러리의 헤더 파일 추가
  2. 라이브러리의 목적 모듈에 속한 함수와 변수 사용
  3. 링커를 통해 실행 파일과 연결
- IDE(통합개발환경) 활용
  - 별도로 링커를 구현할 필요 없이 컴파일 시 자동으로 연속 처리

2. 분리 컴파일

프로그램의 분리

- **클래스의 선언부와 구현부를 분리**하면 프로그램 관리 및 재사용 용이
  - 헤더 파일(.h): 클래스의 선언부 작성
  - 소스 파일(.cpp): 클래스의 구현부 작성
- 헤더 파일의 중복 include를 막기 위해 전처리 지시자를 활용
  - 부분 프로그램에서 중복된 헤더 파일이 include되어도 문제 없이 실행

```
#include 헤더명
#define 헤더명
// 코드 작성
#endif
```

헤더 파일을 중복으로 include해도
한 번만 include한 걸로 처리 가능

2. 분리 컴파일

범위 지정 연산자 ::
접근하려는공간::함수/변수
[예제] 클래스이름::함수/변수
네임스페이스::함수/변수

[예제] 덧셈 프로그램 main.cpp

```
#include "pch.h"
#include <iostream>
using namespace std;

class Adder {
  int op1, op2;
public:
  Adder(int a, int b);
  int process();
};

Adder::Adder(int a, int b) {
  op1 = a; op2 = b;
}

int Adder::process() {
  return op1 + op2;
}
```

```
class Calculator {
public:
  void run();
};

void Calculator::run() {
  cout << "두 개의 수를 입력하세요" << endl;
  int a, b;
  cin >> a >> b;
  Adder adder(a, b);
  cout << adder.process();
}

int main() {
  Calculator calc;
  calc.run();
}
```

실행 결과

```
… 입력하세요
2
3
5
```

2. 분리 컴파일

[예제] 프로그램 분리 → 아래와 같이 선언부와 구현부로 분리

```
#include "pch.h"
#include <iostream>
using namespace std;
```

```
class Adder {
  int op1, op2;
public:
  Adder(int a, int b);
  int process();
};
```

클래스 선언

```
Adder::Adder(int a, int b) {
  op1 = a; op2 = b;
}

int Adder::process() {
  return op1 + op2;
}
```

클래스 구현

```
class Calculator {
public:
  void run();
};
```

클래스 선언

```
void Calculator::run() {
  cout << "두 개의 수를 입력하세요" << endl;
  int a, b;
  cin >> a >> b;
  Adder adder(a, b);
  cout << adder.process();
}
```

클래스 구현

```
int main() {
  Calculator calc;
  calc.run();
}
```

메인 함수

2. 분리 컴파일

[예제] 동일 결과를 얻는 부분 프로그램

```
// Adder.h
#ifndef ADDER_H
#define ADDER_H
class Adder {
  int op1, op2;
public:
  Adder(int a, int b);
  int process();
};
#endif
```

```
// Adder.cpp
#include "pch.h"
#include "Adder.h"
Adder::Adder(int a, int b) {
  op1 = a; op2 = b;
}

int Adder::process() {
  return op1 + op2;
}
```

```
// Calculator.h
#ifndef CALCULATOR_H
#define CALCULATOR_H
class Calculator {
public:
  void run();
};
#endif
```

```
// main.cpp
#include "pch.h"
#include "Calculator.h"

int main() {
  Calculator calc;
  calc.run();
}
```

```
// Calculator.cpp
#include "pch.h"
#include <iostream>
#include "Adder.h"
#include "Calculator.h"
using namespace std;

void Calculator::run() {
  cout << "두 개의 수를 입력하세요" << endl;
  int a, b;
  cin >> a >> b;
  Adder adder(a, b);
  cout << adder.process();
}
```

2. 분리 컴파일

범위 지정 연산자 (Scope resolution operator)

- **접근하려는공간::함수/변수**
- 왼쪽에 아무 것도 없으면 전역 변수를 의미

```
//main.cpp
#include "pch.h"
#include <iostream>

int a = 0; // 전역 변수 a
int main()
{
  int a = 10;
  a = a + 1; // main 내부 지역 변수 a의 값에 +1
  ::a = ::a + 1; // 전역 변수 a의 값에 +1
  std::cout << a << std::endl;
  std::cout << ::a;
  return 0;
}
```

실행 결과

```
11
1
```

2. 분리 컴파일

선언 (Declaration)

- 컴파일러에게 어떤 대상의 **이름**을 알려주는 행위

정의 (Definition)

- 컴파일러에게 어떤 대상의 실제 내용을 알려주는 행위
- 어떤 대상에 대해 대응하는 **메모리 상의 주소**가 정해져야 정의

2. 분리 컴파일

extern 키워드

- 전역 범위로 **선언**되며 다른 파일에서 참조 가능
- 다른 파일에서 선언된 전역 변수를 참조하기 위하여 파일 내에서 extern 키워드를 사용하여 다시 한 번 변수 선언
- 컴파일 시 extern 키워드가 붙은 전역 변수는 다른 파일에 존재하는 변수임을 인식하고 **컴파일 후 링크 시 실제 연결**
- 초기화를 생략하면 0으로 자동 초기화

2. 분리 컴파일

[예제] 선언과 정의

변수 선언과 정의를 동시에: int x;

- 대부분의 변수 선언은 정의와 함께 이뤄지며 메모리 역시 할당 필요

변수 선언만: extern int x;

- 외부 변수 선언 시 다른 파일에서 정의된 변수 이름을 가져와서 사용
- 변수를 위한 메모리 할당은 일어나지 않으므로 정의 없는 선언

함수 선언과 정의를 동시에: int x() {return 0;};

- 함수의 본체를 정의한 경우 함수를 호출할 때 이름에 대응하는 주소 정해짐

함수 선언만: int x();

- 함수 프로토타입 선언: 함수의 이름만 컴파일러에게 알려주는 행위
- 실제로 수행할 내용이 어디 있는지 컴파일러는 모르므로 정의 없는 선언

2. 분리 컴파일

[예제]
선언과 정의

식별자(변수명)
선언 시: 생략
정의 시: 포함(a, x)

```
#include "pch.h"
extern int i; // Declaration without definition
extern float f(float); // Function declaration

float b; // Declaration & definition
float f(float a) { // Definition
return a + 1.0;
}

int i; // Definition
  int h(int x) { // Declaration & definition
    return x + 1;
  }
int main() {
  b = 1.0;
  i = 2;
  f(b);
  h(i);
}
```

2. 분리 컴파일

[예제]
선언과 정의

```
// external.cpp
#include "pch.h"
// 변수 선언
extern int externVariable;
// 함수 정의
void function()
{
  externVariable = 100;
}
```

```
// main.cpp
#include "pch.h"
#include <iostream>
using namespace std;

int externVariable;     // 변수 정의
extern void function(); // 함수 선언
int main()
{
  cout << "exVar:" << externVariable << endl;
  function();
  cout << "exVar:" << externVariable << endl;
}
```

실행 결과

```
exVar: 0
exVar: 1000
```

3. C++ 프로그램 작성

"Hello, World!" 프로그램

- ⟨iostream⟩: ⟨iostream.h⟩로 쓰면 컴파일 에러
- namespace: 함수 이름의 충돌을 막기 위해 정의된 이름 공간
- std에 속하는 cout 객체는 출력 연산자(<<) 사용하여 출력
- std에 속하는 cin 객체는 입력 연산자(<<)를 사용하여 입력
- std에 속하는 endl 함수를 사용하여 줄 바꿈

```
#include "pch.h" // Pre-compiled header
#include <iostream> // Stream declarations
using namespace std;
int main() {
  cout << "Hello, World!" << endl;
}
```

실행 결과

```
Hello, World!
```

3. C++ 프로그램 작성

"Hello, World!" 프로그램

- <cstdlib>: 표준 라이브러리 헤더 중 하나
- system("pause"): 실행된 화면을 보기 위해 일시 정지

```
#include "pch.h"
#include <iostream>
#include <cstdlib>
Using namespace std;

int main()
{
  cout << "Hello World!₩n";
  system("pause");
}
```

실행 결과

```
Hello, World!
```

3. C++ 프로그램 작성

Namespace의 사용

- 같은 이름을 충돌하지 방지하는 문법
- 서로 다른 namespace에 같은 이름의 형식이 존재해도 무방

```
struct Stack
{
    int top;
};

struct Stack
{
    int last;
};
```

컴파일 에러
형식 재정의

```
namespace DemoA
{
  struct Stack
  {
    int top;
  };
}
```

```
namespace DemoB
{
  struct Stack
  {
    int last;
  };
}
```

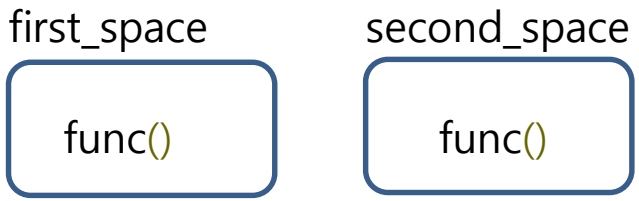
```
DemoA::Stack stacka;
DemoB::Stack stackb;
```

다른 namespace들에 같은 이름 존재 OK

3. C++ 프로그램 작성

Namespace의 사용

- using 지시문을 통해 멤버에 접근 가능




```
#include "pch.h"
#include <iostream>
using namespace std;

// first name space
namespace first_space {
  void func() {
    cout << "Inside first_space" << endl;
  }
}

// second name space
namespace second_space {
  void func() {
    cout << "Inside second_space" << endl;
  }
}
```

```
using namespace second_space;
// 같이 써주면 컴파일 에러
// (함수 호출이 모호합니다)
```

```
using namespace first_space;
int main() {
// This calls function from
// first name space.
  func();
}
```

실행 결과

```
Inside first_space
```

3. C++ 프로그램 작성

Namespace의 사용

- using 지시문을 통해 멤버에 접근 가능
- using 지시문을 과다하게 쓰면 충돌 발생할 수 있음

```
namespace MyNames
{
  struct string {
    string();
  };
}
```

`using namespace std;`를 함께 썼을 때 사용자가 정의한 namespace의 string과 std::string의 충돌이 생겨 컴파일 에러

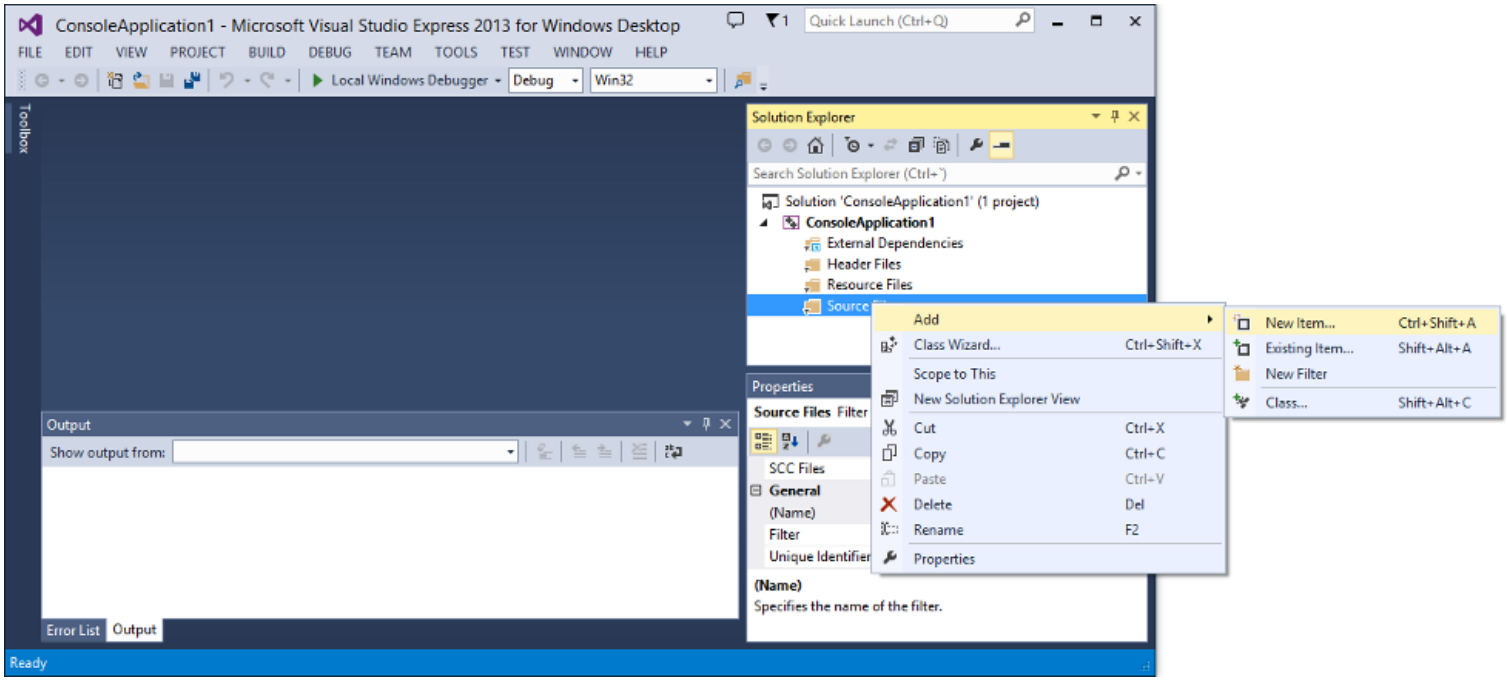
`("string"이(가) 모호합니다)`

3. C++ 프로그램 작성

컴파일러 실행

- IDE(통합개발환경) vs. command-based

```
$ g++ Hello.cpp
$ g++ Hello.cpp -o hello
```



4. iostream 클래스

iostream 출력

- 10진수, 8진수, 16진수(0~9,a~f), 실수로 출력하기
- 제어 문자 (control character): 0~31까지, 27의 경우 escape

```
#include "pch.h"
#include <iostream>
using namespace std;

int main()
{
  cout << "a number in decimal: " << dec << 15 << endl;
  cout << "in octal: " << oct << 15 << endl;
  cout << "in hex: " << hex << 15 << endl;
  cout << "a floating-point number: " << 3.14159 << endl;
  cout << "non-printing char (escape): " << char(3) << endl;
}
```

```
a number in decimal: 15
in octal: 17
in hex: f
a floating-point number: 3.14159
non-printing char (escape):
```

실행 결과

4. iostream 클래스

iostream 출력

- 연쇄 (concatenation) 작성 가능

```
#include "pch.h"
#include <iostream>
using namespace std;

int main() {
  cout << "This is far too long to put on a "
  "single line but it can be broken up with "
  "no ill effects₩nas long as there is no "
  "punctuation separating adjacent character "
  "arrays.₩n";
}
```

실행 결과

```
This is far too long to put on a single line but it can be broken up with no ill effects
as long as there is no punctuation separating adjacent character arrays.
```

4. iostream 클래스

iostream 입력

- 8진수 입력 후 10진수, 16진수로 출력하기
- cout과 유사한 방식으로 cin 사용하여 진법 변환 입/출력 가능

```
#include "pch.h"
#include <iostream>
using namespace std;

int main()
{
  int number;
  cout << "Enter an octal number: ";
  cin >> number;
  cout << "value in decimal = 0" << oct << number << endl;
  cout << "value in hex = 0x" << hex << number << endl;
}
```

```
Enter a decimal number: 17
value in decimal = 015
value in hex = 0xf
```

실행 결과

4. iostream 클래스

외부 프로그램 실행

- 〈cstdlib〉: 표준 라이브러리 헤더 중 하나
- system("실행 파일 경로"): 해당 실행 파일의 결과를 수행

```
#include "pch.h"
#include <iostream>
#include <cstdlib>
using namespace std;

int main()
{
  system("C:/Users/user/source/repos/lec04/Debug/hello.exe");
}
```

실행 결과

```
Hello, World!
```

5. 문자열

[예제] String 클래스 함수

```
#include "pch.h"
#include <string>
#include <iostream>
using namespace std;

int main()
{
  string s1, s2, s3;
  s1.assign("ABC"); // 문자열 할당
  s2.assign(3,'A');
  s3.assign(s1,2,4);
  cout << s1 + " " + s2;
  cout << " " + s3 << endl;

  s1.append("DEF"); // 문자열 결합
  cout << s1 << endl;
  s2.clear();          // 문자열 내용 삭제
  cout << s1.empty() << s2.empty() << endl;

  s1.erase(0, 3);     // 문자열 내용 삭제
  cout << s1 + " " << s1.length() << endl;

  s2.assign(2, 'A'); // 문자열 치환, 비교
  cout << s2.replace(1,2,"B") << endl;
  cout << s2.find("B") << endl;
}
```

실행 결과

```
ABC AAA C
ABCDEF
01
DEF 3
AB
1
```

5. 문자열

[예제] 문자열의 생성, 초기화, 대입, 결합

```
#include "pch.h"
#include <string>
#include <iostream>
using namespace std;

int main()
{
  string s1, s2;      // Empty strings
  string s3 = "Hello, World."; // Initialized
  string s4("I am"); // Also initialized

  s2 = "Today"; // Assigning to a string
  s1 = s3 + " " + s4; // Combining strings
  s1 += " 8 ";  // Appending to a string
  cout << s1 + s2 + "!" << endl;
}
```

실행 결과

```
Hello, World. I am 8 Today!
```

5. 문자열

[예제] 문자열의 결합과 비교

```
#include "pch.h"
#include <string>
#include <iostream>
using namespace std;

int main()
{
  string str1 = "abcde";
  string str2 = "fghij";

  str1 = str1 + str2;

  if (str1 == "abcdefghij")
    cout << "두 문자열은 같다" << endl;
  if (str1 != "123456")
    cout << "두 문자열은 다르다" << endl;
}
```

실행 결과

```
두 문자열은 같다
두 문자열은 다르다
```

6. 파일 읽기 & 쓰기

파일 읽기 & 쓰기

- 〈fstream〉을 사용하여 한 파일을 읽고 다른 파일로 쓸 수 있음

```
#include "pch.h"
#include <string>
#include <fstream>
using namespace std;

int main()
{
  ifstream in("Scopy.cpp");   // Open for reading
  ofstream out("Scopy2.cpp"); // Open for writing
  string s;

  while (getline(in, s)) // Discards newline char
  out << s << "₩n";      // ... must add it back
}
```

6. 파일 읽기 & 쓰기

파일 읽기 & 쓰기

- <fstream>을 사용하여 한 파일의 내용을 한 줄씩 읽을 수 있음

```
#include "pch.h"
#include <string>
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
  ifstream in("Adder.cpp");
  string s, line;
  while (getline(in, line))
  s += line + "₩n";
  cout << s;
}
```

실행 결과

```
// Adder.cpp
#include "pch.h"
#include "Adder.h"
Adder::Adder(int a, int b) {
        op1 = a; op2 = b;
}

int Adder::process() {
        return op1 + op2;
}
```

7. 벡터

C++ 표준 템플릿 라이브러리(STL)

- **템플릿**: 다양한 타입에 적용 가능
- 템플릿 라이브러리에서 제공되는 **함수 활용 가능**
  - **컨테이너**: 자료를 저장하는 클래스 템플릿의 집합
  - **반복자**: 컨테이너의 원소를 순회하는 방법을 추상화한 객체들
  - **알고리즘**: 반복자들을 통해 일련의 작업 수행 가능한 함수들

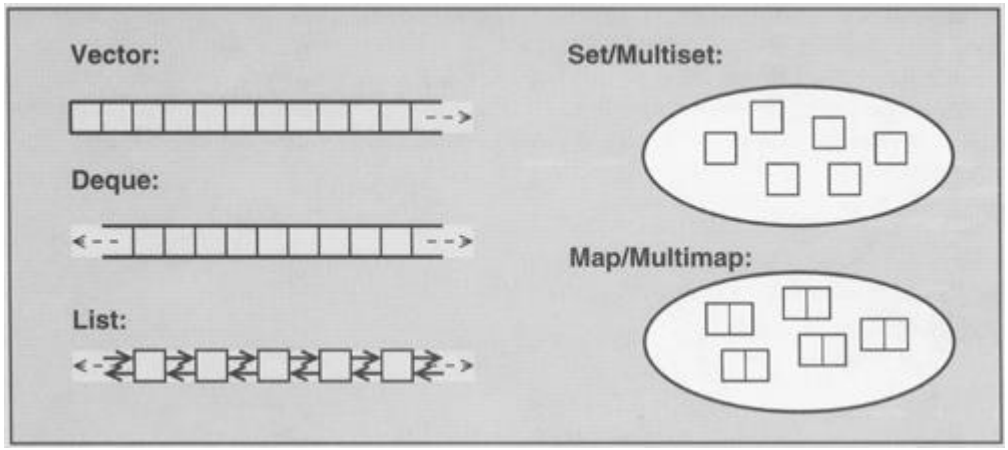
STL의 특징

- 장점: 일반화 지원, 효율성, 이식성 및 확장성 좋음
- 단점: 문법이 달라 가독성 떨어짐, 예외 처리 어려움

7. 벡터

컨테이너: 동일한 요소들을 모아놓은 집합

- **순차 컨테이너**: 동일 객체가 순차적으로 구성된 집
  [예제] 벡터(vector), 리스트(list), 덱(deque) 등
- **연관 컨테이너**: 키(key)를 사용하여 데이터를 찾을 수 있는 집합
  [예제] 집합(set/multiset), 맵(map/multimap) 등



7. 벡터

벡터

- 정의: 사이즈가 유동적인 배열
- 벡터 클래스는 템플릿이기 때문에 다양한 형을 적용 가능
  [예제] vector<shape>
  vector<string>
- **생성자와 연산자**

```
vector<int> v;            // 비어있는 벡터 v 생성
vector<int> v(5);         // 5개의 원소를 가지는 영 벡터 v 생성
vector<int> v1(5,2);      // 2로 초기화 된 5개의 원소를 가지는 벡터 v1 생성
vector<int> v2(v1);       // v1을 복사한 벡터 v2 생성
```

7. 벡터

[예제] 벡터 클래스 멤버 함수 사용

```
#include "pch.h"
#include <vector>
#include <iostream>
using namespace std;

// 벡터 프린트 함수 정의
void print_vec(const vector<int> v)
{
  if (!v.empty())
  {
    for (int i = 0; i < v.size(); i++)
      cout << v[i] << ", ";
    cout << endl;
    }
    else
      cout << "empty vector" << endl;
}
```

```
int main()
{
  vector<int> v(5,2);
  print_vec(v);
  v.pop_back();   // v의 마지막 원소 삭제
  v.push_back(3); // v의 마지막 원소 추가
  print_vec(v);
  cout << v.front() << endl; // 첫번째 원소
  cout << v.back() << endl;  // 마지막 원소
  v.clear();
  print_vec(v);
}
```

실행 결과

```
2, 2, 2, 2, 2,
2, 2, 2, 2, 3,
2
3
empty vector
```

7. 벡터

[예제] 전체 파일을 벡터로 읽기 – 한 줄씩 읽기

```
#include "pch.h"
#include <string>
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;

int main() {
  vector<string> v;
  ifstream in("Adder.cpp");
  string line;
  while (getline(in, line))
    v.push_back(line); // Add the line to the end
  // Add line numbers:
  for (int i = 0; i < v.size(); i++)
    cout << i << ": " << v[i] << endl;
}
```

실행 결과

```
0: // Adder.cpp
1: #include "pch.h"
2: #include "Adder.h"
3: Adder::Adder(int a, int b)
{
4:     op1 = a; op2 = b;
5: }
6:
7: int Adder::process() {
8:     return op1 + op2;
9: }
```

7. 벡터

[예제] 전체 파일을 벡터로 읽기 – 단어 단위 읽기

```
#include "pch.h"
#include <string>
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;

int main() {
  vector<string> words;
  ifstream in("Adder.cpp");
    string word;
  while (in >> word)
    words.push_back(word);
  for (int i = 0; i < words.size(); i++)
    cout << words[i] << endl;
}
```

실행 결과

```
//
Adder.cpp
#include
"pch.h"
#include
"Adder.h"
Adder::Adder(int
a,
int
b)
{
op1
=
a;
op2
=
b;
}
int
Adder::process()
{
return
op1
+
op2;
}
```

7. 벡터

[예제] 벡터 연산

```
#include "pch.h"
#include <iostream>
#include <vector>
using namespace std;

int main() {
  vector<int> v;
  for (int i = 0; i < 10; i++)
    v.push_back(i);
  for (int i = 0; i < v.size(); i++)
    cout << v[i] << ", ";
    cout << endl;
  for (int i = 0; i < v.size(); i++)
    v[i] = v[i] * 10; // Multiplication
  for (int i = 0; i < v.size(); i++)
    cout << v[i] << ", ";
    cout << endl;
}
```

실행 결과

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
0, 10, 20, 30, 40, 50, 60, 70, 80, 90,
```

강의 계획

주차	내용	
~~1~~	~~객체지향 개요 (lecture note)~~	~~개발 환경 구축~~
~~2~~	~~클래스 (lecture note)~~	~~C++ 기초 문법~~
~~3~~	~~객체 (Chap. 1)~~	~~**프로젝트** 팀 구성~~
~~4~~	~~객체 생성과 활용 (Chap. 2)~~	
5	추상화 및 정보 은닉 (Chap. 4, Chap. 5)	
6	상속과 구성 (Chap. 14)	**프로젝트** 제안서 제출
7	다형성과 가상함수 (Chap. 15)	
8	**중간고사**	

질문 및 답변