

# Object-Oriented Programming

Lab #01

Department: 화학공학과

Student ID: 2019101074

Name: 안용상

**\* 코드를 작성하는 문제는 1) 주석(설명)이 포함된 소스코드와 2) 실행 결과를 포함**

1. 컴파일러(compiler)에 대해 설명하라?

바이너리 형식의 기계어 대신,

프로그래밍하기 쉽도록 더 높은 레벨로 추상화한 것이C++과 같은 HighLevel 언어이다

이 HighLevel 언어는 프로그래밍의 편의성을 제공하지만, 컴퓨터가 알아들을 수 있는 형태로 다시 변환하는 과정이 필요함.

이 HighLevel 언어를 컴퓨터가 알아들을 수 있는 LowLevel의 기계어 소스코드로 변환해주는 작업을 "컴파일"이라고 한다

그리고 이 작업을 실행하는 프로그램이 "컴파일러"다.

컴파일러는 일반적으로 "전처리단계"와 "컴파일단계"를 거쳐 소스코드를 컴파일한다.

첫번째는 전 처리 단계이고, 두번째는 컴파일단계이다

컴파일러의 전처리가 소스코드 파일을 읽어 들인다. 그리고 #include나 #define과 같은 전 처리 지시어들을 확인후, 이를 처리하고 전 처리된 소스코드를 임시파일에 저장한다.

두번째는 컴파일단계이다. 컴파일러는 이때 전 처리된 소스코드 임시파일을 읽어 들인다. 그리고 언어의 문법규칙에 따라 어휘분석, 구문분석과 같은 여러 분석과정을 거쳐 기계어코드로 바꾼 후 obj파일을 생성한다

일반적으로 컴파일러와 링커는 따로 존재하지만, 컴파일러 중에는 링커의 역할인 링킹까지 수행하는 경우가 간혹 있다

.

2. C++의 소스 코드(source code)는 어떤 툴들로 작성할 수 있는가?

다양한 IDE(통합개발환경)와 텍스트 에디터를 이용해 작성할 수 있다

대표적으로, IDE로는 마이크로소프트의 Visual Studio가 있다.

Apple에서 개발한 IDE로는 Xcode가 있다

그 외에도 C++개발을 위한 플러그인 지원을 해주는 텍스트 편집기, Windows의 기본

Application인 메모장으로도 작성할 수 있다.

다만 c++코드는 파일확장자를 .cpp로 저장해야 하며, 실행을 위해선 컴파일러가 필요하고, 터미널 환경을 사용해야 하기 때문에, 작성 뿐만 아니라, 컴파일 및 실행까지 일괄적으로 통합해주는 통합개발환경 IDE를 사용하는 것이 좋다.

### 3. 소스코드를 실행가능한 기계어 코드로 변환해 주는 툴(들)을 설명하라.

소스코드를 "실행가능한" 기계어 코드로 바꾸려면, 두가지 과정을 연속적으로 거쳐야 한다.

가장 먼저 C++과 같은 고수준언어나, 어셈블리로 작성된 언어를 이진데이터 코드로 바꿔 (c나 C++ 컴파일러의 경우)obj 파일에 넣어 저장해주는 과정이 필요하다

그리고 그 이진데이터가 저장된 여러 개의 파일들과, 라이브러리들을 결합해서 하나의 실행가능한 기계어 파일로 묶어 생성해주는 과정이 필요하다

첫 번째 과정을, "컴파일러"나 "어셈블러"가 작업하고, 두 번째 과정을 "링커"라는 툴이 작업한다.

### 4. 링커(linker)의 동작에 대해 설명하라?

링킹은 여러 개의 소스코드, 데이터를 모으고 연결해서 메모리에 적재되고, 실행될 수 있는 한 개의 파일로 만드는 작업이다. 링커의 동작과정은 크게 두가지로 나눌 수 있다.

첫째 심볼 재해석 과정 : 어떤 심볼정의를 따를지 결정해두는 작업

컴파일과정을 거쳐 생성된 Object파일들에는 여러가지 정보가 있다. 기계어 코드, 데이터, 그리고 심볼에 대한 정보 등이 포함되어 있다. 심볼들은 변수나 함수의 이름과 같은 것이다. 심볼들은 오브젝트 파일 내에서 정의될 수 있고, 다른 오브젝트 파일에서 가져올 수도 있다. 각 오브젝트 파일에서는 서로 다른 오브젝트 파일에서 정의된 심볼을 참조하는데, 이때 어떤 심볼 정의를 가지고 이를 해석할 것인지를 결정해야 한다. 왜냐하면 여러 오브젝트 파일에 같은 이름으로 함수나 변수가 정의되어 있을 수도 있기 때문에, 중복되는 이름으로 인해 생기는 문제점을 해결하려면 이를 명확하게 결정해 놓아야 한다. 이런 경우 어떤 심볼정의를 따를지 결정해두는 작업이 심볼해석이다.

두번째 재배치 과정

생성된 오브젝트파일은 시작주소가 제로부터 시작하는 코드이다. 하지만 그러한 오브젝트 파일이 여러 개가 중첩되면, 데이터의 주소나 코드의 메모리 참조주소들이 섞여 실행파일에서 본 오브젝트파일에서의 올바른 주소를 매핑하지 못한다. 그래서 이들을 조정하는 작업이 필요하고, 그 작업이 재배치과정이다. 심볼들은 각각의 오브젝트 파일에서 심볼테이블을 통해 전달된다. 심볼테이블은 각 오브젝트 파일에서 해당 심볼이 어느 주소에 위치

하는지에 대한 정보를 가진다. 링커는 여러 오브젝트 파일들의 심볼테이블들의 정보를 하나로 통합한다. 그리고 심볼이 실행파일에 적재되는 주소를 알맞게 매핑해서 저장해둔다.

## 5. 전처리기(preprocessor)의 동작(처리)에 대해 설명하라.

전처리기는 전처리 - 컴파일 - 어셈블 로 이어지는 컴파일링 단계에서 가장 첫번째로 실행되는 부분이다.

#으로 시작해서 \n으로 마무리하는 지시자 코드를 선별해내고, 이에 맞춰 소스파일을 수정해 반영 후, 확장자 .i로 끝나는 임시파일을 생성한 뒤 컴파일러에게 전달한다.

전처리기가 수정한 .i임시파일은 처리하고나서 사라지기때문에 사용자에게는 보이지 않는다.

#으로 시작하는 지시문들에는 include, define, ifdef, ifndef, else, endif등이 포함된다

그럼 전처리문을 왜 쓰는가?

전처리기를 씬으로써 얻는 장점이 있다

첫번째로 코드 가독성을 높일 수 있다.

필요한 함수나 변수를 헤더파일을 통해 미리 정의하고 헤더파일을 불러옴으로써 코드를 간결하게 만들 수 있고, 매크로를 사용해서 반복적인 코드를 단순화 할 수 있다.

두번째로 코드 재사용성이 높아진다.

헤더파일이나 매크로 함수를 사용하는 것만으로 여러 번 쓰이는 반복적인 과정을 대폭 줄이고, 해당 코드를 재사용할 수 있다.

세번째로 코드 유지 보수성을 높일 수 있다.

전처리문에 사용된 부분에 오류가 있을 때 혹은 변경이 필요할 때, 전처리문만 수정하면 유지보수가 가능하도록 만들 수 있다.

이제 실제로 전처리문이 처리되는 방식을 지시문의 종류별로 정리해서 설명해보겠다

### 1. #include

가장 많이 쓰이는 전처리문 지시자이다.

#include "파일" 이나 #include <파일>처럼 작성해서 파일을 참조한다고 명시한다.

전처리기에서 해당파일이 읽어와져서 소스코드에 삽입되고 .i임시파일로 저장된다.

### 2. #define

#define은 매크로를 정의하는 역할을 한다. 전처리기는 해당 매크로를 사용하는 코드

를 매크로로 치환해 코드를 수정한다

예를 들어 <코드> `#define PI 3.141592`라는 매크로 지정문이 있다면, 이는 전처리기에서 다음과 같이 반영된다.

`#define`된 `PI`라는 매크로, 이 매크로가 쓰인 소스코드의 부분들을 추적해서 `PI`에 할당된 값인 `3.141592`라는 value로 치환하고, 코드를 수정한다.

### 3. `#ifdef`, `#ifndef`, `#else`, `#endif`

조건부 컴파일을 수행하는 전처리문이다. 조건에 맞게 소스코드를 컴파일 혹은 무시한다.

이를 다음과 같은 예시를 통해 이해했고, 이를 설명해보겠다.

상황은 다음과 같다. 운영체제에 따라 다른 코드를 사용해야 할 때 조건부 컴파일을 이용하는 상황이다

<코드>

```
#ifdef _WIN32
```

~~~~~(윈도우 운영체제에서 실행되는 코드)

```
#else
```

~~~~~(윈도우 이외의 운영체제에서 실행되는 코드)

```
#endif
```

(if문이 끝남을 알리는 지시문)

또 다른 상황을 예를 들어 설명하겠다

디버그 모드에서만 코드를 실행하고 싶게 만들고 싶은 상황이다

<또다른 예시코드>

```
#ifndef DEBUG
```

~~~~~(ifndef이기 때문에 DEBUG모드일 때 실행되지 않는 코드를 작성한다)

```
#else
```

~~~~~(디버그 모드일 때 실행되는 코드)

```
#endif
```

조건부 전처리문을 사용하면, 컴파일되기전에 상황에 맞지 않는 코드를 배제해버릴

수 있고, 원하는 코드만 실행되게 만들 수 있어서 코드의 이식성을 높일 수 있다(동일한 코드를 다양한 환경에서 실행할 수 있는 성질을 의미한다. 어느 한 플랫폼에 종속되지 않을수록 이식성이 높다.)

#### 4. #pragma

#pragma문은 컴파일러가 코드를 처리하는 방식을 제어할 수 있다

이 외에도 다양한 전처리문이 있다.

#### 6. 목적코드(object code)에 대해 설명하라.

목적코드는 원시코드, 즉 프로그래머가 작성한 소스코드를 컴파일한 결과물이다.

컴파일 과정에서 소스코드 (예를 들면 .cpp)는 전처리를 거쳐 .i로 그리고 다시 컴파일 단계를 거쳐 .h파일로 그리고 컴파일과정의 마지막 단계인 어셈블 단계를 거쳐 코드 파일인 .obj파일로 변환된다.

이때 그 일련의 과정을 거쳐서 "고도로 추상화된 고급언어 소스코드"는 "컴퓨터가 바로 실행할 수 있다는 의미"의 기계어와 매우 밀접한 형태인 이진데이터로 바뀌게 된다.

하지만 목적코드 .obj파일 자체는 완벽한 기계어(컴퓨터가 바로 실행할 수 있는 코드)는 아니고 별도의 추가 작업이 필요하다.

목적코드에는 다음과 같은 정보들이 들어있다.

1. 이진데이터로 변환된 소스코드
2. 심볼테이블
3. 재배치정보
4. 디버그정보

이 중에서 심볼테이블, 재배치정보는 링크단계에서 필요한 정보들이다. 여러 개의 오브젝트 목적파일을 통합하는 링크단계에서, 각 오브젝트 파일이 적절하게 녹아들기 위해서 필수적으로 심볼테이블, 재배치 정보가 필요하다.

목적코드는 아직 실행 불가능한 코드로, 링크단계에서 링킹되기 위해 필요한 작업들을 거쳐 링크단계에 들어가기 직전의 코드이다. 링킹 단계를 거치기 위해 필요한 모든 전처리를 거친 코드라고 해도 무방하다. 그렇게 목적코드를 만드는 이유는, 링킹 단계에서 원활하게 실행파일로 변환되게 만들기 위해서이다

다음은 목적코드의 필요성에 대해서 말해보겠다.

우선 목적코드와 같은 실행파일과 원시 소스코드 중간 단계에 속하는 코드파일이 없다면,

실행파일로 바로 컴파일이 될 텐데, 이렇게 되면 디버깅이나 최적화가 어렵다. 그래서 이를 보완한다.

## 7. 정적 라이브러리(static library)와 동적 라이브러리(dynamic library)에 대해 설명하라.

라이브러리는 재사용이 필요한 기능을 미리 만들어, 언제 어디서든 참조해 사용한다는 목적을 가진다.

이런 라이브러리는 정적, 동적 라이브러리로 분류된다.

### # 정적라이브러리

정적라이브러리는 실행파일에 직접 저장되는 방식의 라이브러리다. 라이브러리에 작성된 소스코드는 목적파일로 번역되며, 링킹을 통해 하나의 실행파일 내에 소속되게 된다.

즉 실행파일이 생성되는 컴파일타임에 실행파일 내에 복사되는 라이브러리다.

특징은, 실행파일 내에 복사되어 이미 저장되어 있기 때문에, 실행파일을 가지고 있다면, 라이브러리를 따로 갖고 있지 않아도 된다. 또 실행할 때는 실행파일 내부에 이미 다 저장되어 있기 때문에 외부에서 참조하지 않아도 돼서 빠르다는 장점이 있다.

다만 프로그램 크기가 비대해지고, 같은 라이브러리가 메모리에 이미 있어도 상관없이 올리기 때문에, 메모리를 많이 차지하게 된다는 단점이 있다. 또한 라이브러리 내용을 수정하기 어렵다는 단점이 있다.

정적라이브러리 코드는 실행파일이 실행되어서 프로세스가 생성될 때 메모리에 할당된다. 정적라이브러리 코드는 실행파일의 코드와 똑같이 CODE영역에 할당된다. 또 해당 정적 라이브러리가 수행될 때는 Stack영역에 지역변수를 할당하면서 수행된다.

### #동적라이브러리

동적라이브러리는 런타임에 연결된다.

동적라이브러리는 각각의 프로세스마다 메모리에 라이브러리의 내용을 할당하는 방식을 따르지 않는다.

라이브러리를 한 번만 할당해서, 이를 사용하는 여러 프로세스가 공유해서 사용한다.

그래서 object파일을 만들면서 라이브러리 내용이 아닌 라이브러리가 위치한 주소만을 가지고 있다가, 실행되는 시점인 런타임에, 그리고 그 라이브러리가 메모리에 위치될 때, 명시된 주소로 가서 필요한 것들만 참조해오는 방식을 따른다.

이 동적라이브러리를 사용하면, 프로그램 자체에 라이브러리를 복사해 넣지않고, 주소만

을 가지고 있기 때문에, 자연히 프로그램의 사이즈가 작아지고, 라이브러리가 한번만 메모리에 올라가면, 그것을 또 올리는 일없이 공유해서 사용한다는 점 때문에 메모리공간을 적게 차지한다는 장점이 있다. 하지만 프로그램 내부에서 바로 가져오는 것이 아니라, 프로그램 외부에서 파일로 존재하는 라이브러리를 참조해 메모리에 올려 가져와야 한다는 점에서 실행시간에 영향을 미칠 수 있다. 다시 말해 실행시간이 다소 느려질 수 있다.

다만 핵심적인 장점으로, 라이브러리가 외부에 위치하기 때문에, 라이브러리를 손쉽게 수정하고 바로 반영되게 할 수 있다는 것이다. 다만 함수의 인수나 반환 값이 변경되었다면 다시 링크해야 한다.

동적라이브러리를 사용하는 방식을 다시 둘로 분류할 수 있는데

암시적 연결과 명시적연결이 그것이다.

암시적 연결은 프로그램이 실행될 때 DLL이 로드되는 방식이고

명시적연결은 프로그램 실행 중에, 그리고 DLL을 로드 하라는 명령이 있을 때 로드 된다.

암시적 연결은 임포트 라이브러리가 필요하며

명시적연결은 함수와 속한 DLL의 이름을 명시하기에 임포트 라이브러리가 필요하지 않다.

#### 8. C++ 프로그램에서 반드시 필요한 함수의 이름과 동작에 대해 설명하라.

C++프로그램에서 반드시 필요한 함수는 main함수이다

main함수는 C++프로그램의 시작점이다. 프로그램의 실행을 처음 여는 함수이다.

main함수는 프로그램에서 운영체제로부터 단 한번만 호출된다,

main함수의 반환 값은 정수형 int이다 그래서 int main이라고 명시한다

그리고 반환 값은 관습적으로 프로그램이 성공적으로 종료 했는지의 여부를 나타낸다. 일반적으로 0을 반환하면 정상종료라는 뜻이다.

main함수는 내부에서 프로그램이 실행될 때 필요한 코드들이 작성되고, 매개변수를 받을 수 있다.

#### 9. 자신의 이름을 출력하는 프로그램을 작성하라.

<코드>

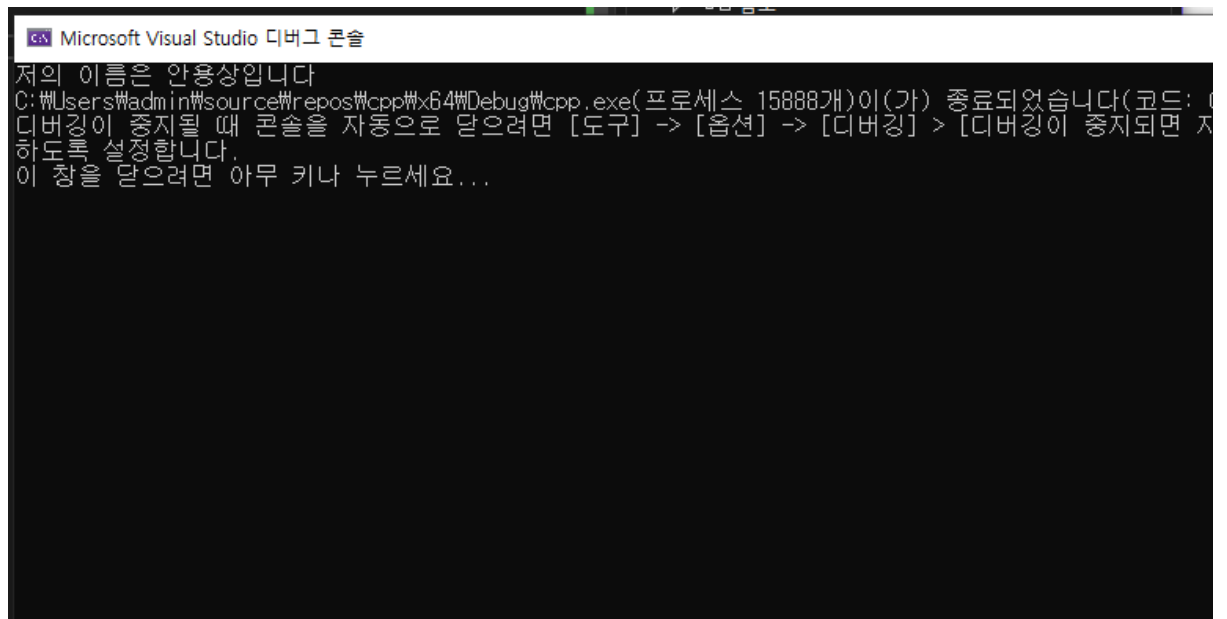
```
#include <stdio.h> //printf를 쓰기위해 stdio.h라이브러리를 헤더파일에 올려놓기
#define myname "안용상" //myname 매크로 정의
int main() {
    printf("저의 이름은 %s입니다", myname); //myname변수를 포맷해서 출력하게 만들
    return 0;
}
```

<터미널 결과>

저의 이름은 안용상입니다

C:\Users\qhfkd\Project1\Wx64\Debug\Project1.exe(프로세스 39700개)이(가) 종료되었습니다(코드: 0개).

이 창을 닫으려면 아무 키나 누르세요...



10. 아래의 출력이 동일한 결과를 출력하는지를 확인하고, 그 결과가 나오는 이유를 설명하라.

```
std::cout << 6 << '\n';  
std::cout << "6" << '\n';
```

<결과>

6

6

C:\Users\qhfkd\Project1\Wx64\Debug\Project1.exe(프로세스 35180개)이(가) 종료되었습니다(코드: 0개).

이 창을 닫으려면 아무 키나 누르세요...

<이유>

두 코드는 출력결과가 동일했다. 왜냐하면 정수형 데이터로써의 6과 개행 문자나 문자열 6과 개행문자나 서로 다른 데이터 형식을 출력하지만, 출력결과에서는 숫자 6과 문자 6이 모두 같은 문자로 보여지므로 출력결과가 같게 나온다.



하지만 사실 둘은 출력하는 내용이 다르다.

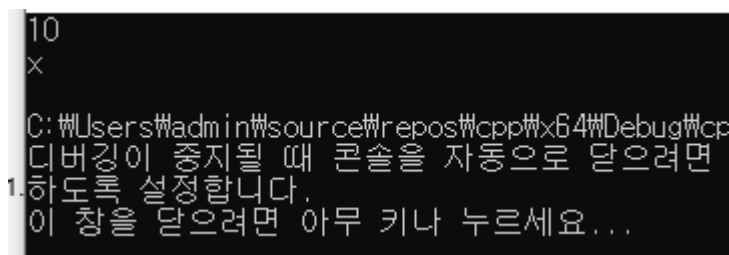
먼저 첫번째 줄의 코드는 `int`형 타입의 데이터 6을 출력하고 개행문자를 출력한다. 정수형 숫자들은 `c++` 컴파일러에 의해서 출력 시 문자열로 변환되어 출력된다.

두번째 코드에서는 6이라는 데이터를 `char`데이터타입으로 출력한다. 코드상에서 보이지는 않지만 문자열의 마지막에 `'\0'`이라는 널문자가 존재하며, `char`데이터타입의 경우 `'\0'` 널문자를 만날 때까지 출력을 계속한다. 이 경우 `'6'`이라는 문자열과 보이지는 않지만 존재하는 `'\0'` 널문자가 출력 6을 만들어 냈다. 그리고 다시 `cout` 명령을 `'\n'` 개행문자에 적용해서 행을 새로 열게 된다.

결과적으로 첫번째 코드와 두번째 코드의 출력은 동일했지만, 내부적으로 동작하는 과정은 서로 상이함을 알 수 있다

11. 아래의 출력이 동일한 결과를 출력하는지를 확인하고, 그 결과가 나오는 이유를 설명하라.

```
int x = 10;
std::cout << x << '\n';
std::cout << "x" << '\n';
```



```
10
x

C:\Users\admin\source\repos\cpp\64\Debug\cp
디버깅이 중지될 때 콘솔을 자동으로 닫으려면
1. 하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

출력은 서로 달랐다

첫번째 `cout`에서는 변수로서의 `x`를 출력했기 때문에 변수 `x`를 찾아 그 값을 출력했다

하지만 두번째 `cout`에서는 하나의 영문자 `x`를 출력했기 때문에, 변수 `x`와는 독립적으로 영문자 `x`를 출력했다.

12. 음이 아닌 정수에 대해 설명하라.

0과 자연수를 뜻하는 음이 아닌 정수는 다시 말해 양의 정수와 0이다. 음이 아닌 정수는 일반적으로 자연수를 다루는 용도로 많이 쓰인다.

`C++`에서는 `unsigned int`, `unsigned long` 등을 이용해 음이 아닌 정수를 표현할 수 있다.

13. 변수의 선언(declaration)과 정의(definition)에 대해 설명하라.

선언과 정의의 차이는 메모리를 할당하느냐의 여부이다

선언은 그 실체에 대한 메모리를 할당하지 않으며, 컴파일러에게 변수의 정보만을 준다.

즉 변수의 실체에 대한 메모리를 갖지 않으며, 해당 변수의 이름, 데이터 타입, 메모리 크기 등의 정보만을 전달한다.

정의는 선언과 동시에 실체에 대한 메모리를 할당한다. 그리고 컴파일러에게 실제 변수를 생성하도록 한다.

변수의 선언과 정의를 나눠 놓은 이유는 다음과 같다

여러 소스파일에서 변수를 공유할 경우가 있을 수 있다.

이때 한 소스파일에서 정의한 변수를 다른 소스파일에서 사용할 때, 그 다른 소스파일에 서 또 정의를 할 게 아니라, 선언만 해서 사용할 수 있게 하기위해서이다.

이는 몇가지 장점을 갖는다

변수의 이름충돌을 방지하고, 재사용성과 유지보수성이 증가하고, 변수의 정의를 공유하므로 한번만 정의하게 되어 불필요한 메모리 로스를 줄인다.

14. 할당(대입, assignment) 연산자에 대해 설명하라.

할당연산자는 "="를 말하며, 변수에 값을 대입하는 역할을 한다.

반드시 왼쪽 피연산자에 오른쪽 피연산자의 값이 할당된다

왼쪽 피연산자는 반드시 변수여야하고, 오른쪽 피연산자는 변수, 상수, 수식이 되어야 한다.

또 왼쪽 피연산자와 오른쪽 피연산자가 자료형이 다르다면 자동으로 형변환을 시켜준다.

15. 아래에서 식별어(identifier)로 사용이 불가능한 것을 찾아서 이유를 설명하라.

`x, if, 2x, x2, public, _1, _if, two&`

---

정답 : `if, 2x, public, two&`이다

`2x` : 숫자는 가장 첫번째에 나오는 경우를 제외한다고 하기 때문에 `2x`는 안된다

`If, public` : `if`는 c++에서 조건문을 나타내는 키워드이고, `public`은 c++의 예약어이기 때문에 안된다

`Two&` : 식별자는 알파벳, 숫자, 밑줄(\_)로만 구성될 수 있고, 특수문자(&)는 사용할 수 없다.

16. float 타입과 double 타입의 차이에 대해서 설명하라.

float타입과 double타입은 공통점이 둘 다 부동 소수점 숫자를 표현하기 위해 있는 c++ 기본 데이터타입이라는 것이다

둘을 비교한 테이블을 만들어 보았다

|           | Float           | double                |
|-----------|-----------------|-----------------------|
| 크기        | 32비트 (4바이트)     | 64비트(8바이트)            |
| 유효숫자(정밀도) | 7자리             | 15자리                  |
| 장점        | 속도나 메모리 효율이 좋음  | 높은 정밀도                |
| 단점        | double보다 낮은 정밀도 | 큰 메모리 필요로 함, 연산 속도 느림 |

17.  $1.25 \times 10^{-12}$  을 C++의 상수로 표현하라.

1.25e-12

18. 부동소수점(floating-point) 표현에서 정밀도(precision)에 대해 설명하라.

부동소수점 표현에서 정밀도(precision)는 소수점 이하의 자릿수를 나타내며, 이 값이 클수록 더 정확한 값을 나타낼 수 있다.

32비트 부동소수점 표현에서는 대략 7자리의 정밀도가 있으며, 이진수로 표현되기 때문에 소수점 이하의 이진수 자릿수에 대한 정밀도를 고려해야 한다.

부동소수점의 정밀도는 계산의 정확성과 관련이 있으며, 정확한 계산이 필요한 분야에서는 적절한 자료형과 연산 방법을 선택해야 한다.

19. 아래의 차이에 대해 설명하라.

5, '5', "5"

>>>5는 10진수로 표현된 정수형 상수

>>>'5'는 ASCII 코드에서 53을 나타내는 문자형 상수

>>>"5"는 문자열 상수이며 char 배열로 처리된다. 메모리에서는 연속된 문자 '5'와 '\0' (null 문자)로 표현된다

20. 아래의 출력 결과와 이유를 설명하라.

```
int x;  
x = 'A';  
std::cout << x << " " << x+1 << std::endl;
```

```
std::cout << (char)x << " " << (char)(x+1) << std::endl;
```



먼저 65와 66이 출력된 이유는 다음과 같다

Int형 x가 있을 때 문자 "A"를 할당 받았다면 아스키표준에 의해서 "A"의 아스키코드 값으로 바뀌어 정수로 저장된다

따라서 x를 출력하면 "A"의 아스키코드 값인 65가 x+1은 66이 출력이 된다

다시 (char)x 즉 c스타일 캐스팅에 의해서 character로 int x가 바뀌게 되어 다시 아스키 코드 값으로 변환된다

따라서 char(x)는 char(65) 이고 다시 "A"로 변환되고 char(x+1)은 char(66) 이므로 "B"로 변환된다.

21. 아래의 두가지 문자의 차이를 설명하라.

'n', '\n'

>> '\n'은 이스케이프 문자면서, 개행문자로, 줄 바꿈의 역할을 한다. 반면 'n'은 하나의 문자 'n'을 나타낸다.

22. unsigned char의 용도에 대해 설명하라.

Unsigned char는 0부터 255까지 값을 저장하는 데이터 타입이다.

1바이트의 크기를 가진다. 그래서 256가지의 표현을 할 수 있고, unsigned라서 음수가 없다.

Unsigned char의 용도는 무엇일까?

바이너리 데이터를 처리할 때 사용하는 용도이다

이미지나 영상에서 각 픽셀은 색상정보를 0-255까지의 숫자로써 저장한다. 이때 unsigned char의 범위와 완전히 맞아 떨어지기 때문에, 불필요한 메모리 낭비없이 딱 들어맞게 색상정보를 표현할 수 있다.

또한 네트워크의 경우에도 바이트단위로 데이터를 전송하기에 unsigned char가 자주 쓰인다.

23. 아래의 연산자들의 연산 우선 순위를 높은 것부터 낮아지는 순으로 정렬하라.

`+, -, *, /, %, =`

높은 순으로

1순위 : `*` `/` `%`

2순위 : `+` `-`

3순위 : `=`

24. 두 개의 정수를 사용자로부터 입력을 받아서, 덧셈, 뺄셈, 곱셈, 나눗셈의 몫(quotient)과 나머지(remainder)를 출력하는 프로그램을 작성하라.

<코드>

```
#include <iostream>

int main() {
    int a, b;
    std::cout << "두 정수를 입력하십시오: "; //사용자를 위한 문구 출력
    std::cin >> a >> b; //a와 b에 차례로 std::cin명령어 적용
    int sum = a + b; //합을 정수데이터 타입의 변수에 담는다
    int sub = a - b; // 뺄셈 결과를 정수데이터 타입의 변수에 담는다
    int 곱 = a * b; // 곱셈 결과를 정수데이터 타입의 변수에 담는다
    int 몫 = a / b; // 나눗셈 몫을 정수데이터 타입의 변수에 담는다
    int 나머지 = a % b; // 나눗셈 나머지를 정수데이터 타입의 변수에 담는다

    std::cout << "덧셈: " << sum << std::endl; // sum변수를 출력하고 줄바꿈
    std::cout << "뺄셈: " << sub << std::endl; //sub 변수를 출력하고 줄바꿈
    std::cout << "곱셈: " << 곱 << std::endl; // 곱 변수를 출력하고 줄바꿈
    std::cout << "나눗셈 몫: " << 몫 << std::endl; // 몫 변수를 출력하고 줄바꿈
    std::cout << "나눗셈 나머지: " << 나머지 << std::endl;
    // 나머지 변수를 출력하고 줄바꿈

    return 0;
}
```

}

<출력결과>

두 정수를 입력하십시오: 125

365

덧셈: 490

뺄셈: -240

곱셈: 45625

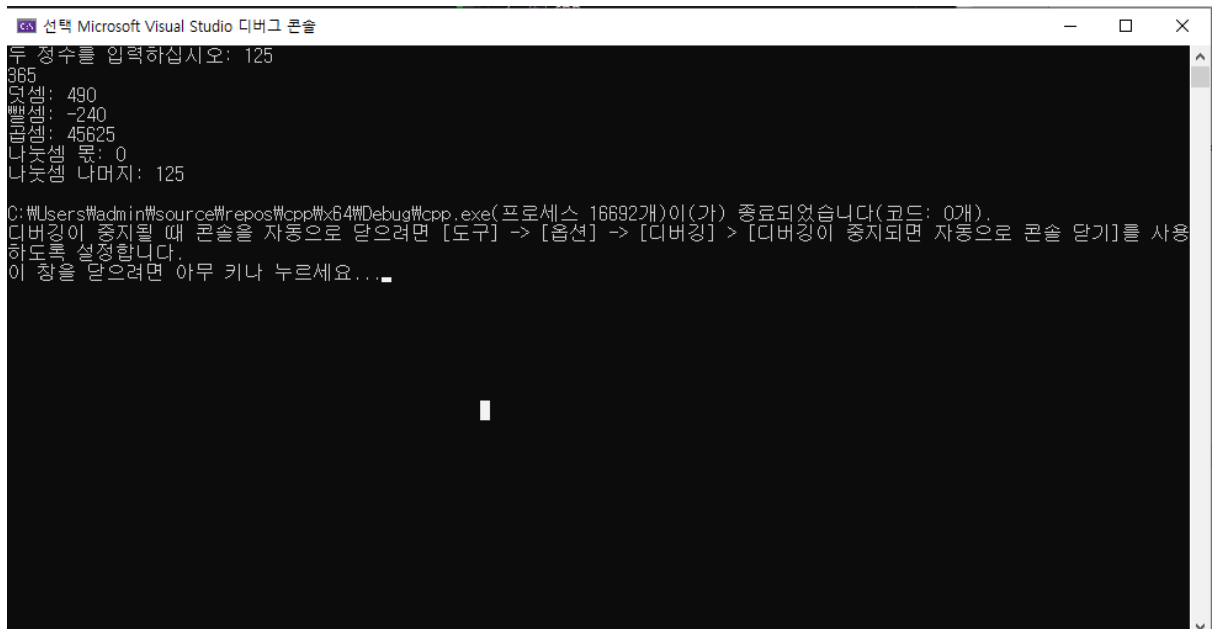
나눗셈 몫: 0

나눗셈 나머지: 125

C:\Users\Wadmin\source\repos\cpp\64\Debug\cpp.exe(프로세스 16692개)이(가) 종료되었습니다(코드: 0개).

디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.

이 창을 닫으려면 아무 키나 누르세요...



```
선택 Microsoft Visual Studio 디버그 콘솔
두 정수를 입력하십시오: 125
365
덧셈: 490
뺄셈: -240
곱셈: 45625
나눗셈 몫: 0
나눗셈 나머지: 125
C:\Users\Wadmin\source\repos\cpp\64\Debug\cpp.exe( 프로세스 16692개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
█
```

25. 두 개의 실수를 사용자로부터 입력을 받아서, 덧셈, 뺄셈, 곱셈, 나눗셈의 결과를 출력하는 프로그램을 작성하라.

<코드>

```
#include <iostream>

int main() {
    double x, y; //x와 y를 선언
    std::cout << "첫번째 숫자 입력: ";
    std::cin >> x; // x에 사용자 입력값 할당
    std::cout << "두번째 숫자 입력: ";
    std::cin >> y; // y에 사용자 입력값 할당

    std::cout << x << " + " << y << " = " << x + y << std::endl;
    std::cout << x << " - " << y << " = " << x - y << std::endl;
    std::cout << x << " * " << y << " = " << x * y << std::endl;
    std::cout << x << " / " << y << " = " << x / y << std::endl;

    return 0;
}
```

<출력결과>

첫번째 숫자 입력: 165

두번째 숫자 입력: 87

165 + 87 = 252

165 - 87 = 78

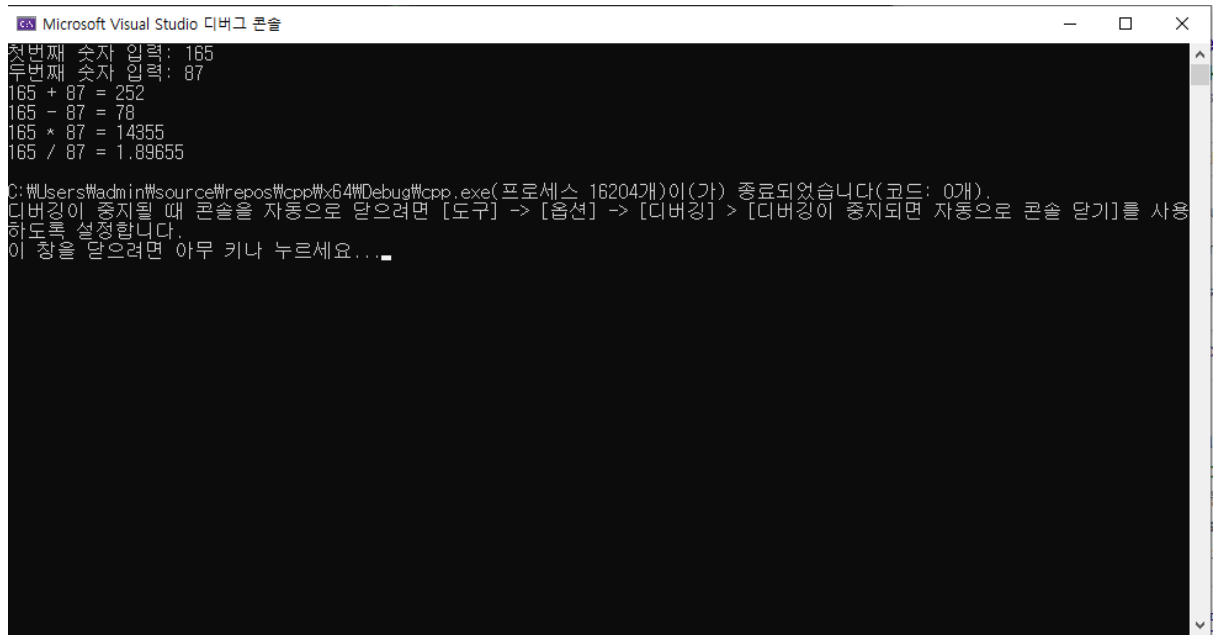
165 \* 87 = 14355

165 / 87 = 1.89655

C:\Users\Wadmin\source\repos\cpp\wx64\Debug\cpp.exe(프로세스 16920개)이(가) 종료되었습니다(코드: 0개).

디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.

이 창을 닫으려면 아무 키나 누르세요...



26. 아래 코드의 출력 결과를 확인하고, 그 결과가 나오는 이유를 설명하라.

```
int x = 5;  
double r = 5;  
char c = 'A';  
unsigned char u1 = 250, u2;  
std::cout << x/2;  
std::cout << r/2;  
std::cout << (x+2.)/2;  
std::cout << (r+2)/2;  
std::cout << x/(1+1.);  
std::cout << r/(1+1);  
std::cout << static_cast<char>(c+2);  
std::cout << (c+2);  
std::cout << static_cast<int>(u2 = u1+5);  
std::cout << static_cast<int>(u2 = u1+15);
```

1. `std::cout << x/2;`

출력결과 : 2

정수형 타입이고 5를 할당받은 x를 다시 정수인 2로 나누면 결과값도 정수로 나와서 2가 반환된다

2. `std::cout << r/2;`

출력결과 2.5

double형 타입이며 5를 할당받은 r을 정수로 나누면, 컴퓨터는 자동으로 int를 double로 변환하여 계산을 수행하기에, 출력결과도 double이고, 2.5를 반환받는다



3. `std::cout << (x+2.)/2;`

출력결과 3.5

`int`형 타입이고 5를 할당받은 `x`에 `.2`라는 실수를 더해주면, 자동으로 실수형으로 바뀌어 계산된다. 따라서 `(x + .2)` 부분은 7.이 되고 이를 다시 2로 나누면, 실수인 3.5가 출력된다

4. `std::cout << (r+2)/2;`

출력결과 3.5

`double`형 타입이며 5를 할당받은 `r`에 정수 2를 더하면 정수 2는 `double`로 바뀌고 다시 이를 2로 나누면 `double`형태로 결과가 나와서 3.5가 나온다

5. `std::cout << x/(1+1.);`

출력결과 2.5

먼저 분모인 `(1+1.)` 부분에서 최종적으로 이부분은 실수2.으로 바뀌게 되고, 정수형 타입이며 5를 할당 받은 `x`를 실수형 분모로 나누게 되므로 결과 또한 실수로 출력되어 2.5가 된다

6. `std::cout << r/(1+1);`

출력결과 2.5

분모는 정수 2이지만, 분모가 `double`형태여서 출력결과도 `double`이다

따라서 2.5가 출력된다

7. `std::cout << static_cast<char>(c+2);`

출력결과 c

`c`는 `charactor`문자로 "A"를 할당받았다. 그래서 `(c+2)`를 수행하면 `c`는 "A"라는 문자에 해당하는 아스키코드 값이 된다. 그래서 `c+2`, 즉 "A" +2는 63+2로 65가 되지만, `static_cast<char>`에 의해서 해당 아스키코드값에 해당하는 문자인 "C"를 출력한다.

8. `std::cout << (c+2);`

출력결과 67

`c+2`는 "A"+2와 같다. 문자 더하기 정수와 같은 상황에서 문자는 아스키코드로 변환되어 숫자와 연산을 한뒤 숫자를 반환한다

따라서 `c+2`는 `"A"+2` 이고, 다시 이것은 `65+2`로 `67`이 된다.

```
9. std::cout << static_cast<int>(u2 = u1+5);
```

출력결과 255

`unsigned char u1 = 250, u2;`라고 처음에 주어졌다.

`Unsigned char`는 0부터 255까지의 값을 가지며, 이 범위를 벗어날 경우 256으로 나눈 나머지가 대입된다

여기서 `u2`는 `u1`에 5를 더한값이므로 255이고 0부터 255라는 범위안에 들기 때문에, 나머지연산이 필요가없다. `u2`는 따라서 255의 `unsigned char`이다 하지만 `static_cast<int>`에 의해 255는 그대로 정수가된다

```
10. std::cout << static_cast<int>(u2 = u1+15);
```

출력결과 9

`u2`는 265가된다. 이때 0-255범위인 `unsigned char`는 나머지를 계산한다.

`265%256`은 9이므로

9가 출력되었다

27. 블록 주석이 중첩해서 사용할 수 없음을 예를 들어 설명하라.

예를 들어서

```
/*저는 컴퓨터에 관심이 많아 컴퓨터 공학과 수업을 듣고 있는 */화학공학과 안용상*/이라고 합니다*/
```

라는 중첩된 블록주석 구문이 있다고 하자.

이 경우 가장 먼저 주석을 열었던 `/*` 부분이 가장 첫번째로 나오는 `*/`블록주석을 닫는 기호를 자신의 짝 블록주석으로 인식하게 되고, 두번째 블록주석을 여는 기호를 그냥 문자로 취급해버린다.

따라서 주석은

```
/*저는 컴퓨터 공학과에 관심이 많아 컴퓨터 공학과 수업을 듣고 있는 */화학공학과 안용상*/
```

이 부분만 처리가 되고,

```
이라고 합니다*/
```

이 부분은 그냥 어느 것에도 감싸지지 않은 노출된 형태로 존재해, 컴파일 오류를 불러일으킨다.

따라서 블록주석은 중첩해서 사용할 수 없다.

## 28. 아래 코드의 문제점에 대해서 설명하라

```
#include <iostream>
int main() {
    double C, r = 0;
    const double PI = 3.14159;
    // Formula for the area of a circle given its radius
    C = 2*PI*r;
    // Get the radius from the user
    std::cout << "Please enter the circle's radius: ";
    std::cin >> r;
    // Print the circumference
    std::cout << "Circumference is " << C << '\n';
}
```

---

여기서 문제점은 다음과 같다

원의 둘레를 구하는 로직이 ( $C=2*PI*r$ 부분) 아직  $r$ 에 사용자의 입력을 받기 전 상태에 나와버려서, 사용자의  $r$ 입력을 반영하지 못하고 초기화된 값인 0을 반영해서 항상  $C=0$ 이 나오게 된다. 이는 원의 둘레를 구하려는 프로그램 제작의 목적에 맞지 않는다

```
std::cout << "Please enter the circle's radius: ";
std::cin >> r;
```

이 부분을  $C = 2*PI*r$ ;이 부분의 바로 위로 올리면

사용자의 입력을 받아 원의 둘레를 구하는 프로그램을 목적에 맞게 문제없이 만들어낼 수 있다.

## 29. $x=4$ 이고, $y=5$ 인 경우의 아래 수식의 연산 결과는? ( $x, y, z$ 의 값)

1.  $z = x++ + y$ ;

=>9이다 / 위 수식이 끝나고  $x$ 에 ++연산이 들어갔으므로  $4+5$ 의 값인 9가 나온다

2.  $z = x++ + --y$ ;

=>8이다 /  $x$ 는 위 수식이 끝난후 ++가 적용되었고,  $y$ 는 --가 적용된 이후 수식에 반영되었다. 따라서  $4 + (5-1)$ 의 값으로 8이 나온다.

3.  $z = ++x-y--$ ;

=>0이다 /  $x$ 는 먼저 1을 더한 후 위 수식에 반영되었고,  $y$ 는 위 수식이 끝난후 --가 적용되었으므로,  $(4+1)-5$ 의 결과로 0이 나온다

4. `z = + + x+1;`

=> 5이다 / + +는 ++와 같지 않다. 단항 연산자 +를 두번한 것이므로 연산 결과는 x+1과 같다.

5. `z = +-x;`

=> -4이다 / 단항연산자 +와 -를 연달아쓴 결과 x의 부호를 바꿔준 값이 z에 넣어진다.

30. `x = 2945` 일때, 아래의 연산 결과에 대해서 설명하라.

1. `x%10`

2945를 10으로 나눈 나머지를 구하는 것이므로 이경우 1의자리 숫자인 5가 결과로 반환된다

2. `x/10%10`

c++에서는 /과 %이 왼쪽부터 실행되며, 같은 연산자 우선순위를 가진다 따라서

`x/10`을 먼저 진행한 값인 294를 이어서 `%10`연산을 거쳐 `294%10` 을 해서 최종결과는 4가 나온다.

3. `x/100%10`

2번에서 설명한대로 /와 %는 같은 연산자 우선순위이다. 따라서 왼쪽부터 실행하면

`x/100 => 29`

`29%10 => 9`

따라서 최종적으로 9가 반환된다.

31. `float`형 실수의 정수의 일의 자리를 `int`형 변수에 저장하고 출력하는 프로그램을 작성하라.

<코드>

```
#include <iostream>
#include <stdio.h>
int main() {
    float f; // float f를 선언
    int i; // int i를 선언
    std::cout << "실수를 입력해주세요" << std::endl;
    std::cin >> f;
    // while문으로 데이터타입에 맞는 입력이 있을 때까지 계속 입력을 지우고, 다시 입력을 받는다
    while (std::cin.fail() == 1) {
```

```

        printf("실수를 입력해주세요");
        std::cin.clear();
        std::cin.ignore(10000, '\n');
        std::cin >> f;
    }
//일의 자리숫자를 뽑기 위해 f를 정수로 바꾼 뒤, 10으로 나눈 나머지를 구한다.
    int _10으로_나눈_나머지 = (int)f%10;
    printf("일의 자리 숫자는 : %d",_10으로_나눈_나머지);
}

```

<출력>

실수를 입력해주세요

d

실수를 입력해주세요d

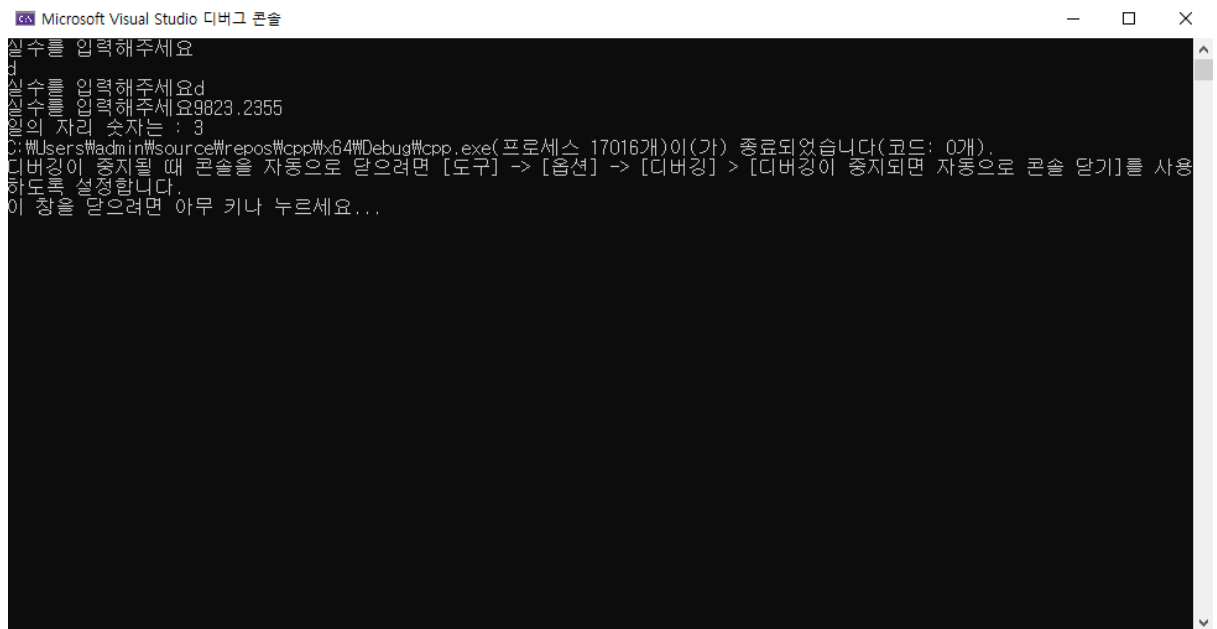
실수를 입력해주세요9823.2355

일의 자리 숫자는 : 3

C:\Users\Wadmin\source\repos\cpp\64\Debug\cpp.exe(프로세스 17016개)이(가) 종료되었습니다(코드: 0개).

디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.

이 창을 닫으려면 아무 키나 누르세요...



여기서 코드설명

Std::cin에서 사용자가 실수로 혹은 고의로 실수를 입력하지 않았을 때, 예를들면 문자를 입력했을 때, 원하는 코드흐름을 방해할 소지가 있다.

따라서 그런 불확실성을 원천 차단하고자, std::cin.fail()이라는 데이터타입에 맞는 값이 들

어왔는지를 검증해주는 부가기능을 추가했다

그리고 while문을 넣어서 std::cin.fail()이 1이면 즉, 데이터타입에 맞지 않는 값이 들어오면, 들어올때 까지, 입력버퍼를 지우고 다시 입력을 받도록 구현했다.

32. 사용자로부터 직사각형의 폭과 높이를 입력을 받아서, 면적과 둘레를 계산해서 출력하는 프로그램을 작성하라.

<코드>

```
#include <iostream>
#include <stdio.h>

int main() {
    int width, height;
    std::cout << "직사각형 width 를 입력하라(m)\n";
    std::cin >> width;
    // while문으로 데이터타입에 맞는 입력이 있을 때까지 계속 입력을 지우고, 다시 입력을 받는다

    while (std::cin.fail()==1) {
        std::cout << "정수를 입력해주세요" << std::endl;
        std::cin.clear(); // 비정상적인 이전 상태를 초기화하고 새로운 입력을 받을 수 있도록 함
        std::cin.ignore(10000, '\n'); //입력버퍼를 지운다.
        //'\n'개행문자가 나올 때 까지 10000개의 문자까지 지운다.
        std::cin >> width;
    }

    printf("당신이 입력한 직사각형의 너비(width)는 %d미터 입니다\n\n", width);

    std::cout << "직사각형의 height를 입력하라(m)\n";
    std::cin >> height;
    // while문으로 데이터타입에 맞는 입력이 있을 때까지 계속 입력을 지우고, 다시 입력을 받는다

    while (std::cin.fail() == 1) {
        printf("정수를 입력하세요!");
        std::cin.clear(); // 비정상적인 이전 상태를 초기화하고 새로운 입력을 받을 수 있도록 함

        std::cin.ignore(10000, '\n'); //입력버퍼를 지운다.
        //'\n'개행문자가 나올 때 까지 10000개의 문자까지 지운다.

        std::cin >> height;
    }
    printf("당신이 입력한 직사각형의 높이(height)는 %d미터 입니다\n\n", height);

    printf("직사각형 둘레 \n직사각형 너비 : %d \n직사각형 높이 : %d\n2x(%d + %d) = %d미터\n-----\n", width, height, width, height, 2*(width+height));
```

```
int area = width * height;
printf("직사각형 넓이 %d\n직사각형 너비 : %d\n직사각형 높이 : %d\n%d x %d
= %d제곱미터", width, height,width,height, area);
}
```

<출력>

직사각형 width 를 입력하라(m)

123

당신이 입력한 직사각형의 너비(width)는 123미터 입니다

직사각형의 height를 입력하라(m)

543

당신이 입력한 직사각형의 높이(height)는 543미터 입니다

직사각형 둘레

직사각형 너비 : 123

직사각형 높이 : 543

$2 \times (123 + 543) = 1332$ 미터

-----

직사각형 넓이

직사각형 너비 : 123

직사각형 높이 : 543

$123 \times 543 = 66789$ 제곱미터

C:\Users\admin\source\repos\cpp\wx64\Debug\cpp.exe(프로세스 2628개)이(가) 종료  
되었습니다(코드: 0개).

이 창을 닫으려면 아무 키나 누르세요...

```
Microsoft Visual Studio 디버그 콘솔
직사각형 width 를 입력하라(m)
123
당신이 입력한 직사각형의 너비(width)는 123미터 입니다

직사각형의 height를 입력하라(m)
543
당신이 입력한 직사각형의 높이(height)는 543미터 입니다

직사각형 둘레
직사각형 너비 : 123
직사각형 높이 : 543
2x(123 + 543) = 1332미터

직사각형 넓이
직사각형 너비 : 123
직사각형 높이 : 543
123 x 543 = 66789제곱미터
C:\Users\admin\source\repos\cpp\64\Debug\cpp.exe( 프로세스 2628개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...■
```

## 여기서 코드설명

Std::cin에서 사용자가 실수로 혹은 고의로 정수를 입력하지 않았을 때, 예를들면 문자를 입력했을 때, 원하는 코드흐름을 방해할 소지가 있다.

따라서 그런 불확실성을 원천 차단하고자, std::cin.fail()이라는 데이터타입에 맞는 값이 들어왔는지를 검증해주는 부가기능을 추가했다

그리고 while문을 넣어서 std::cin.fail()이 1이면 즉, 데이터타입에 맞지 않는 값이 들어오면, 들어올때 까지, 입력버퍼를 지우고 다시 입력을 받도록 구현했다.

그리고 기본 basis단위는 가장 기본적인 단위인 미터로 했다.