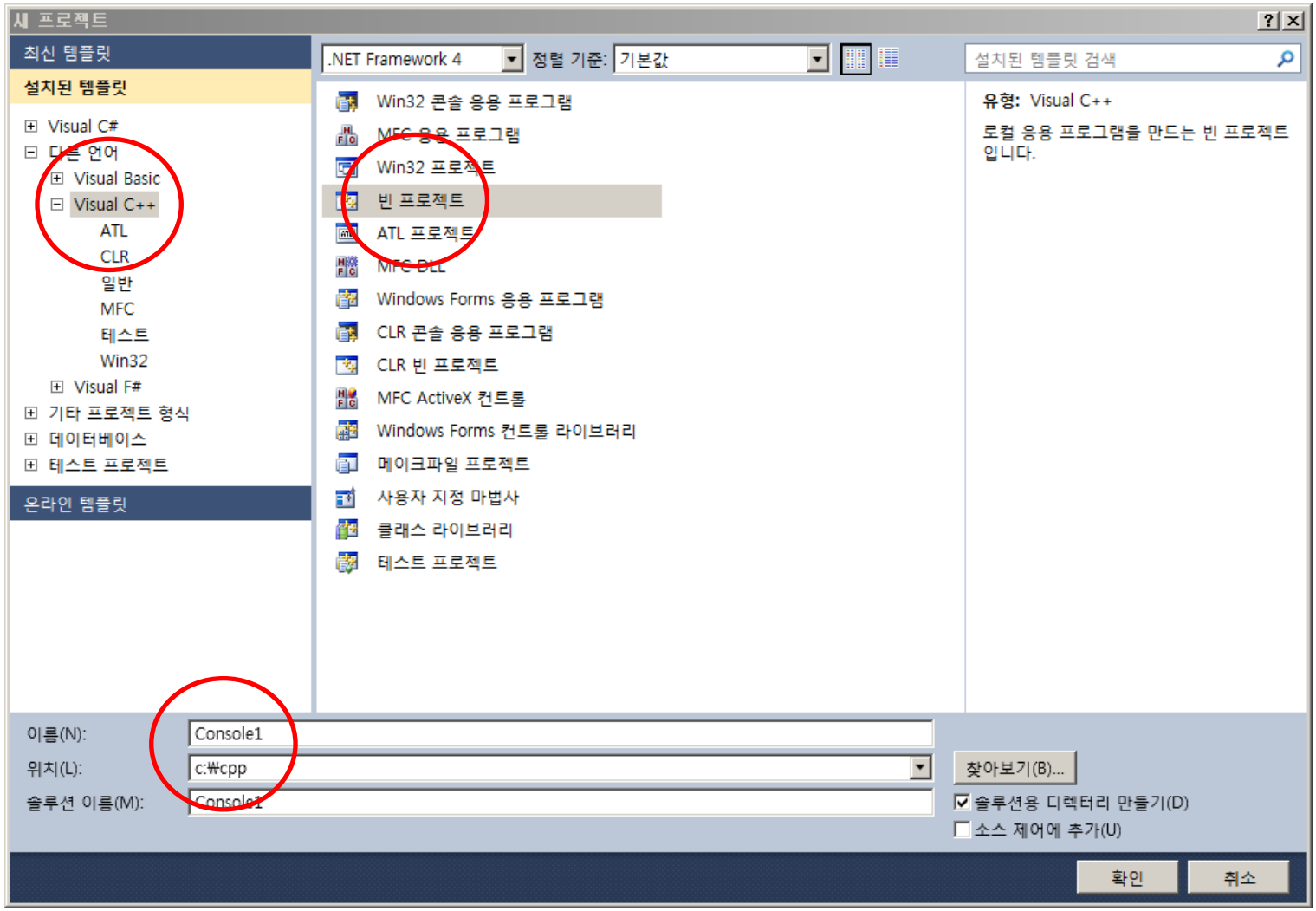

AlphaGo
VS

2장 OOP를 공부하는 이유

변영철 교수
(ycb@jejunu.ac.kr)

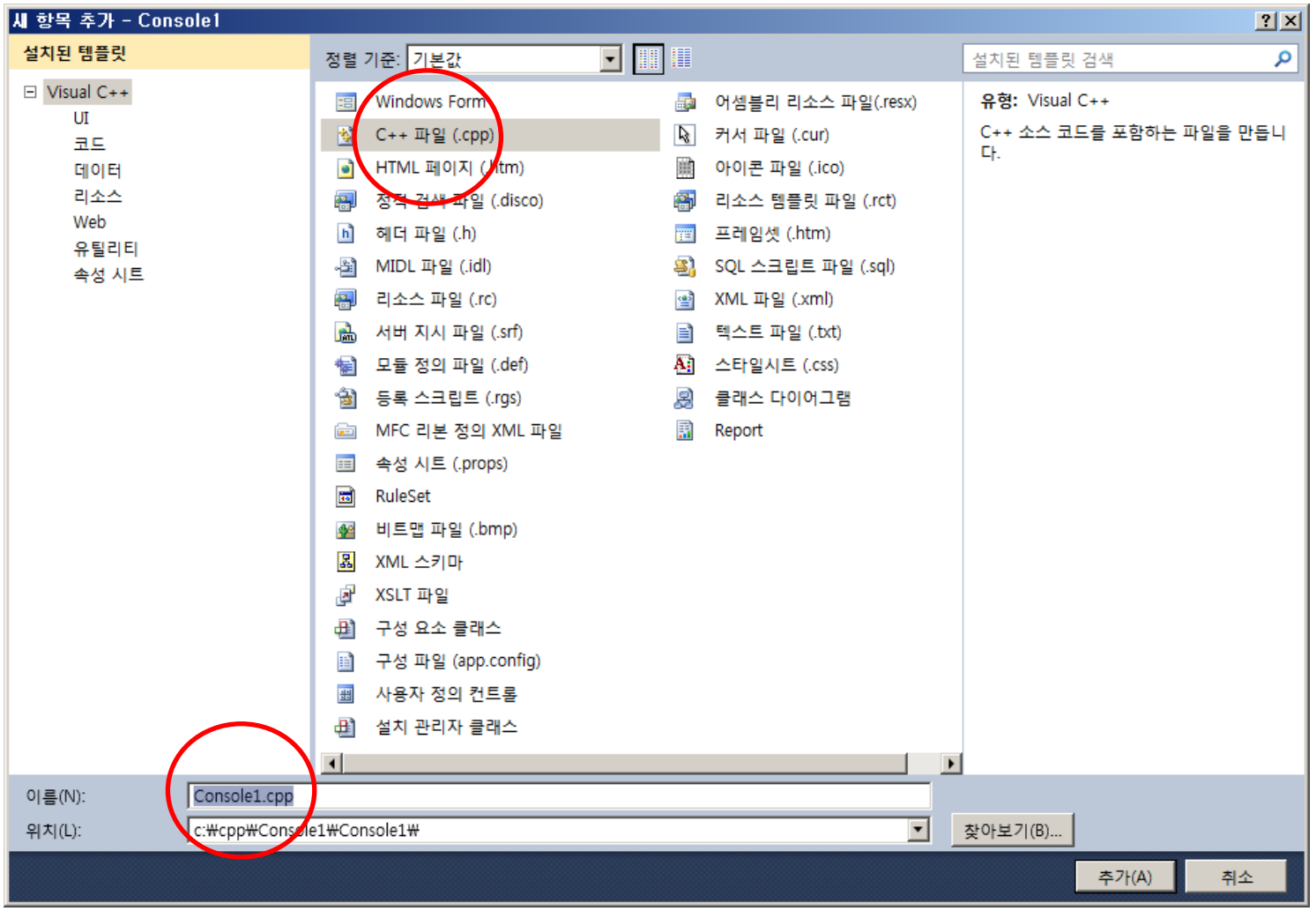
1. 아주 간단한 C 프로그램

- 파일 > 새로 만들기 > 프로젝트 메뉴 선택



1. 아주 간단한 C 프로그램

- 프로젝트 > 새 항목 추가



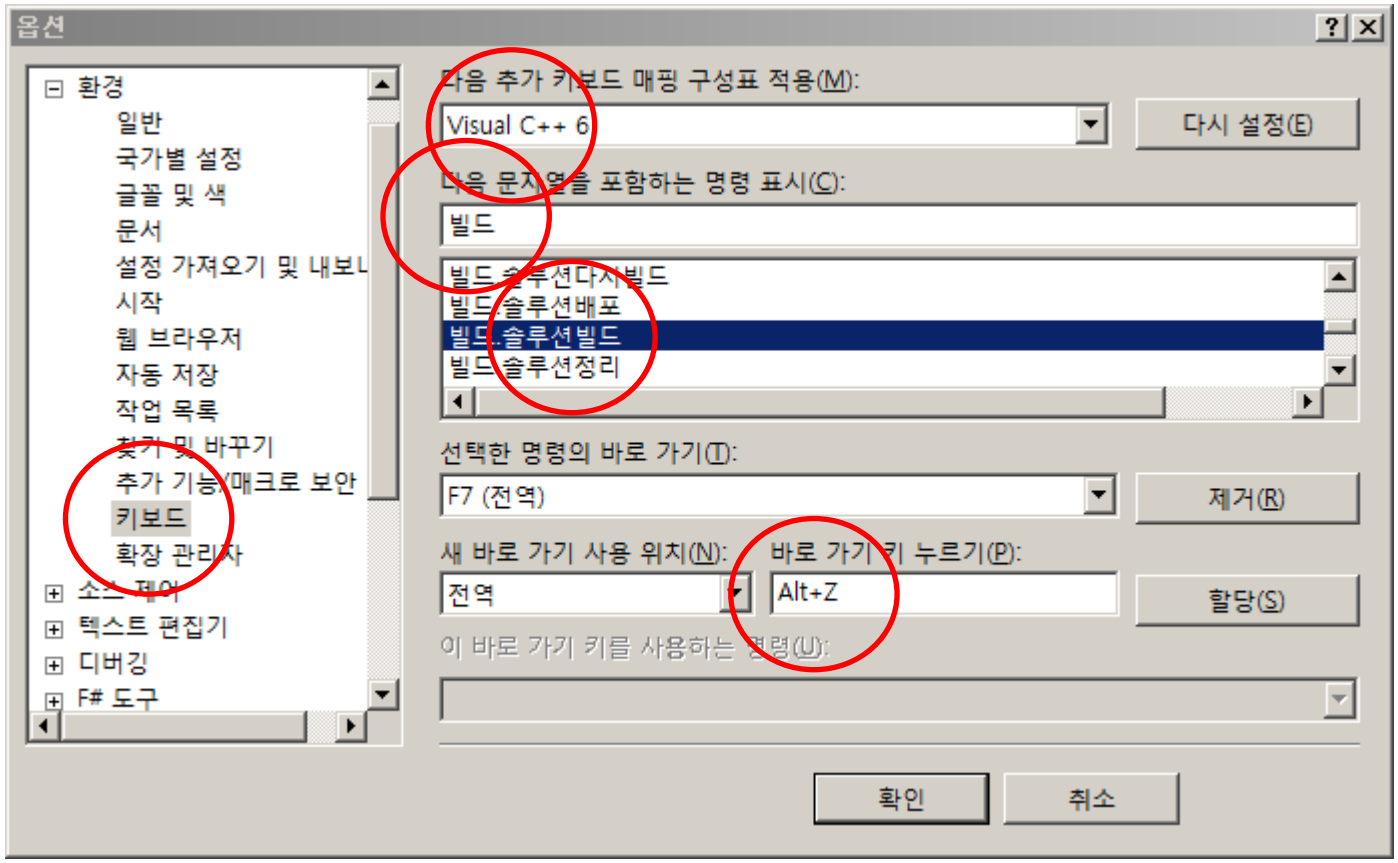
1. 아주 간단한 C 프로그램

- **코드 입력**

```
//-------------------------
// Console1.cpp   (c) Y. Byun
//-------------------------

#include <stdio.h>

void main()
{
    printf("Hello, World!₩n");
}
```

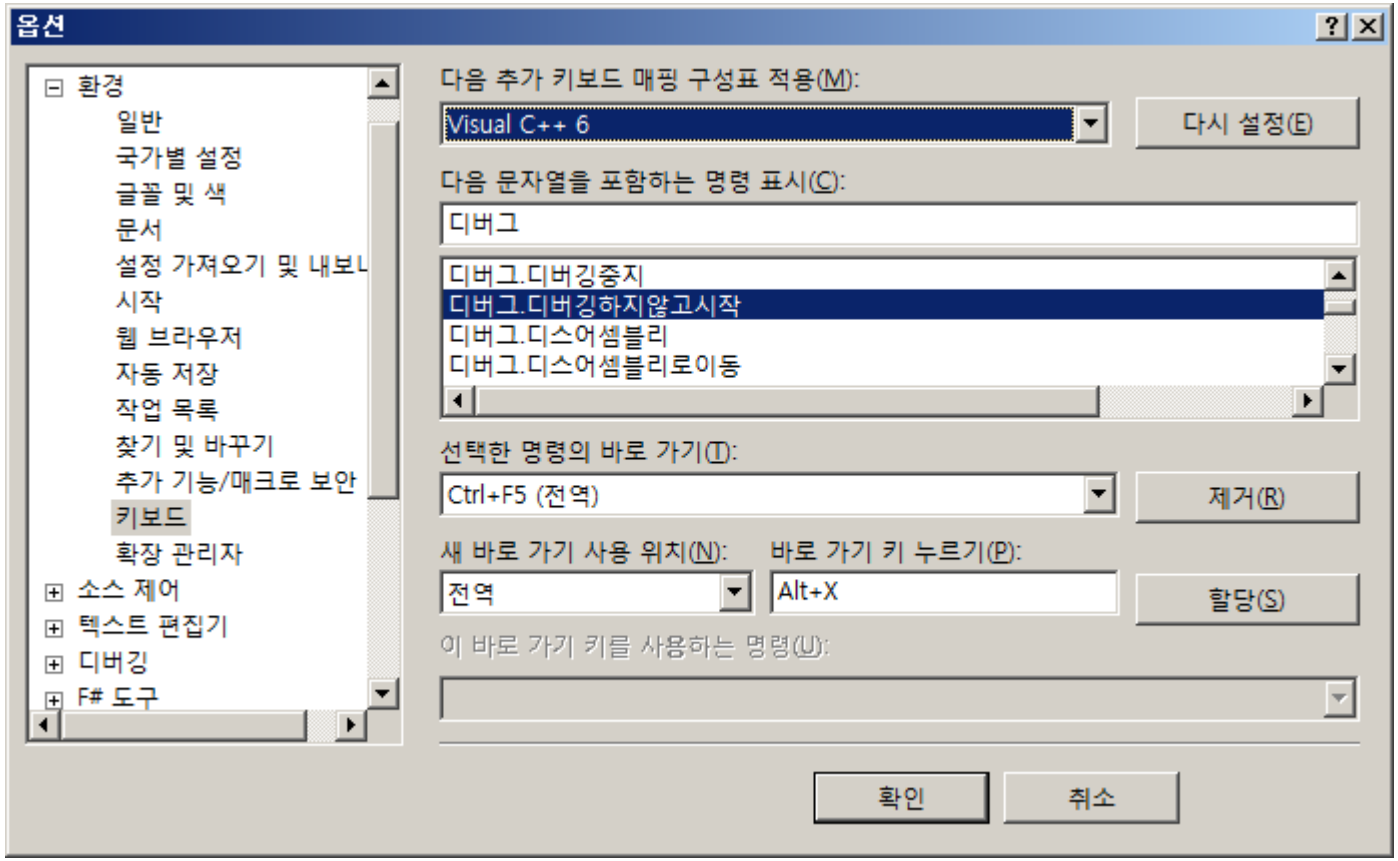
1. 아주 간단한 C 프로그램

- 컴파일 및 실행
- 단축키 지정 (컴파일) : 도구 > 옵션 메뉴 선택



1. 아주 간단한 C 프로그램

- 단축키 지정(실행)



2. 조금 복잡한 C 프로그램

- 구조화 프로그램
  - 실행 순서가 항상 위에서 아래로만 진행
  - 프로그램을 쉽게 읽을 수(이해할 수) 있음 : 가독성(readability)이 좋아짐.
  - 구조화 프로그래밍 언어: C, Java 등
- 모듈
  - 함수나 클래스를 모듈이라고 함.
  - 심지어는 '{'과 '}'에 의해 둘러싸인 블록
  - 프로그램 파일(예를 들어 a.cpp)
  - C언어를 모듈별 분할 컴파일이 가능한 언어라고 하는데, 여기에서의 모듈은 하나의 프로그램 파일을 의미

2. 조금 복잡한 C 프로그램

- 아래와 같이 2개의 지역변수를 정의하자.

```
#include <stdio.h>

void main()
{
        int iX;
        int iY;

        iX = 2;
        iY = 3;

        int iResult = iX + iY;

        printf("두 개의 값을 더한 결과: %d₩n", iResult);
}
```

2. 조금 복잡한 C 프로그램

- **지역변수를 전역변수로 바꾸기**

```
#include <stdio.h>

int iX;
int iY;

void main()
{
        iX = 2;
        iY = 3;

        int iResult = iX + iY;

        printf("두 개의 값을 더한 결과: %d₩n", iResult);
}
```

2. 조금 복잡한 C 프로그램

- 변수 정의와 선언
  - 정의(definition) : 메모리에 변수를 만듦
  - 선언(declaration) : 컴파일러에게 변수 정보를 알림

대한독립을 만방에 ?? 하노니...

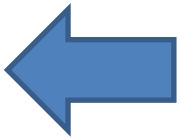
2. 조금 복잡한 C 프로그램

- 지역변수와 전역변수 특성
  - 변수 생명주기(life time) : 전역 변수는 프로그램이 실행될 때 만들어지고 프로그램이 종료될 때 사라짐. 이에 반해 지역 변수는 함수가 실행될 때 만들어지고 함수가 끝날 때 사라짐
  - 변수 사용 범위(scope) : 전역 변수는 모든 함수에서 접근할 수 있음. 이에 반해 지역 변수는 함수 내에서만 접근할 수 있음
- 지역변수를 전역변수로 바꾸는 이유
  - 중요한 값을 저장하고 오래 유지되어야 하는 변수들, 여러 함수에서 사용되는 변수들은 전역 변수로 정의

3절. 함수를 이용한 프로그램

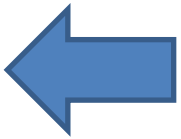
- 수백 줄로 작성된 프로그램은?
  - 보기만 해도 머리가 아프다?!
- 추상화(abstraction)
  - 복잡한 내용을 간단하게 줄여서 표현하는 것
  - 예) 어제 무엇을 했나요?
- 코드 추상화
  - 복잡한 코드를 간단히 표현하는 것
  - 코드 추상화 하기 : Assign, Add

```c
#include <stdio.h>

int iX;
int iY;

void Assign (int x, int y)
{
    iX = x;
    iY = y;
}

void main()
{
    int iResult;

    Assign(2, 3);

    iResult = iX + iY;

    printf("두 개의 값을 더한 결과: %d₩n", iResult);
}
```



코드 두 줄을 Assign 함수로 추상화한 것

```c
#include <stdio.h>

int iX;
int iY;

void Assign(int x, int y)
{
    iX = x;
    iY = y;
}

int Add()
{
    return iX + iY;
}

void main()
{
    int iResult;

    Assign(2, 3);

    iResult = Add();

    printf("두 개의 값을 더한 결과: %d₩n", iResult);
}
```



코드 한 줄을 Add 함수로 추상화한 것

3절. 함수를 이용한 프로그램

- 코드 추상화의 장점
  - 보기에 편해졌다.
  - 쉽게 읽을 수 있다.
  - 그 결과 프로그램 분석이 쉬다.
  - 고치기 쉽다(프로그램 유지 보수).
  - 작성한 함수들을 나중에 쉽게 재사용 할 수 있다.
  - 따라서 프로그램 생산성도 높아진다.

4절. 100개의 변수와 1000개의 함수

- 하지만! 앞 프로그램은…
  - 함수를 이용한 잘 모듈화된 구조화 프로그램이긴 하지만,
  - 이해하는 데에도 별 문제가 없지만,
  - 프로그램 크기가 커져서 100 개의 전역 변수와 1000개의 함수로 구성된 프로그램의 경우에는?

4절. 100개의 변수와 1000개의 함수

- 예상해볼 수 있는 문제점들
  - 오류가 발생할 경우 오류 수정이 어려움
  - 잘못된 접근을 방지할 수 있는 장치가 없음
  - 코드 재사용이 어려움
  - 결국 소프트웨어 위기(crisis)가 발생

4절. 100개의 변수와 1000개의 함수

- 왜 문제점이 발생할까?
  - 어떤 함수가 어떤 변수를 액세스하는지 쉽게 알 수 없기 때문
  - 어떤 함수가 어떤 함수를 호출하는지 쉽게 알 수 없기 때문
  - 함수와 변수들이 기준 없이 뒤섞여 있어서 재사용 시 어디까지 사용해야 하는지 쉽게 알 수 없기 때문
  - 초기화 함수와 같이 반드시 호출해야 하는 함수를 쉽게 알 수 없기 때문

4절. 100개의 변수와 1000개의 함수

- 해결 방법
  - 이와 같은 문제점을 근본적으로 소프트웨어 위기를 극복할 수 있는 방법
  - OOP 방법
  - 구조화 프로그램을 객체지향 프로그램(OOP)으로 바꿈으로써 이러한 문제점들을 해결할 수 있음
  - OOP를 공부해야 하는 필연적인 이유

그래서 OOP를 공부한다!