제1장 기초 사항

INFINITYBOOKS
인피니티북스

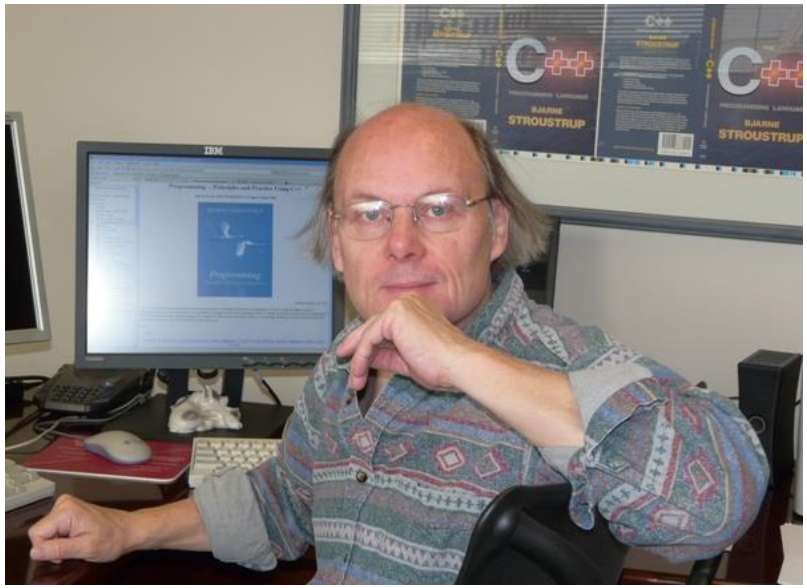
이번 장에서 학습할 내용

- C++ 언어의 역사
- C++ 언어의 특징
- 객체 지향의 간단한 소개
- C++ 개발 과정
- 첫번째 예제 설명
- 변수와 상수
- 연산자

C++에 대한 기초적인 사항들을 살펴봅니다.

C++ 언어의 개발

- C++는 1980년대 초에 AT&T 벨연구소의 Bjarne Stroustrup에 의하여 개발
- C++는 C언어를 유지, 확장한 것
- C with Classes -> C++
- C++는 C언어에 클래스 개념을 추가하고 이어서 가상 함수, 연산자 중복 정의, 다중 상속, 템플릿, 예외 처리 등이 기능이 차례로 추가



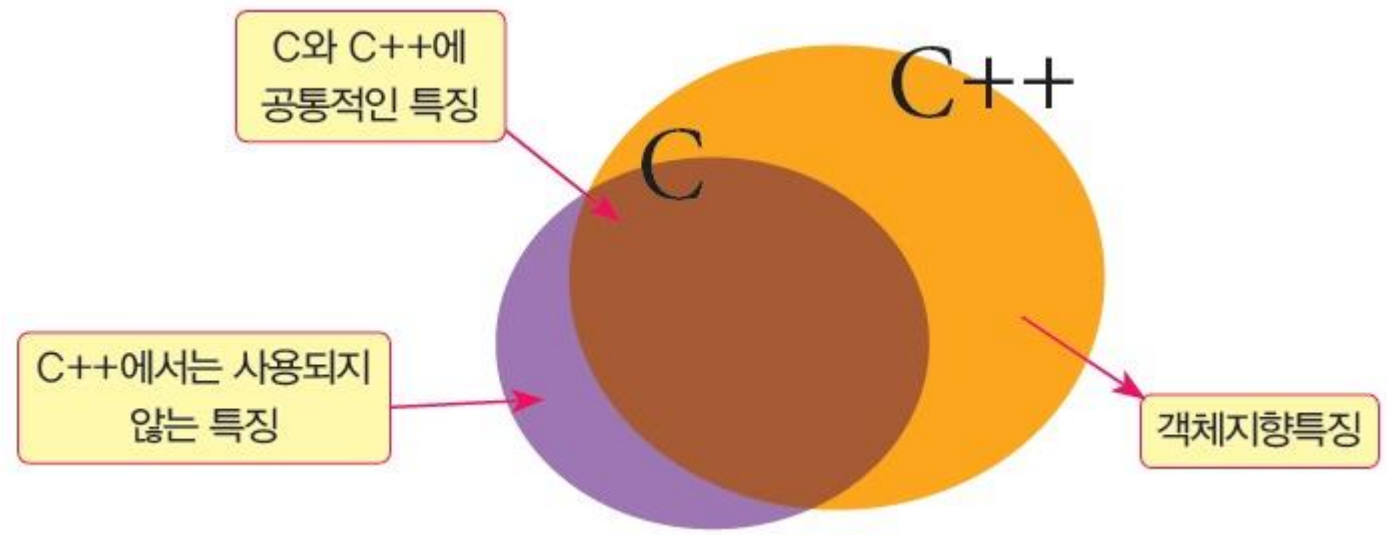
C++ 버전

연도	C++ 표준	비공식적인 이름
1998	ISO/IEC 14882:1998	C++98
2003	ISO/IEC 14882:2003	C++03
2011	ISO/IEC 14882:2011	C++11
2014	ISO/IEC 14882:2014	C++14
2017	아직 정해지지 않음	C++17
2020	아직 정해지지 않음	C++20

C++의 설계 철학

- 엄격한 타입 검사, 범용 언어, 효율적, 이식성
- 여러 가지의 프로그래밍 스타일을 지원 (절차 지향 프로그래밍, 데이터 추상화, 객체 지향 프로그래밍, 일반화 프로그래밍)
- 프로그래머가 자유롭게 선택할 수 있도록 설계
- 최대한 C와 호환
- 플랫폼에 의존적이거나 일반적이지 않은 특징은 제거

C++ 특징

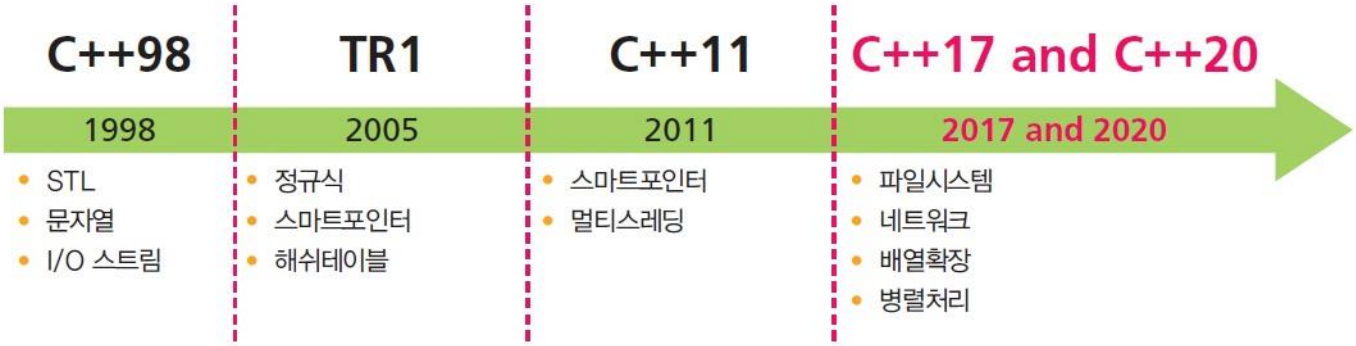


C++ 특징

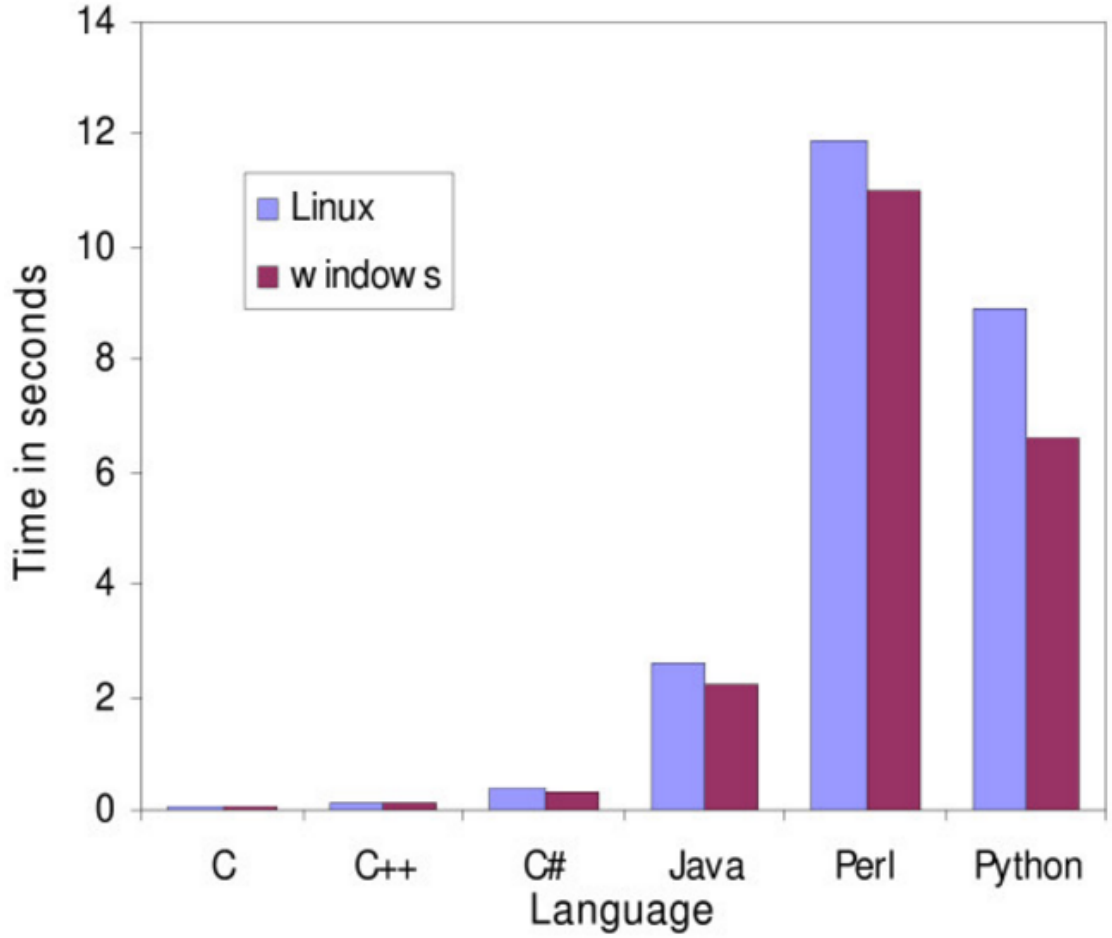
- **클래스(class) -** 클래스를 이용하여 하나의 객체의 속성과 동작들을 한곳으로 모아서 정의할 수 있다.
- **상속(inheritance) -** 클래스를 상속받아서 기존의 코드를 재사용할 수 있다.
- **연산자 중복(operator overloading)** - 대상에 따라서 동일한 연산자로 새로운 연산을 정의할 수 있다.
- **함수 중복(function overloading)** - 매개 변수만 다르면 동일한 이름의 함수를 여러 개 만들 수 있다.
- **new와 delete 연산자** - 동적 메모리 할당과 해제를 담당하는 연산자이다.
- **제네릭(generics)** - 클래스 정의를 자료형에 상관없이 재사용하는 기술이다.

모던 C++

- C++ 11과 C++ (14)는, 많은 새로운 기능을 도입
  - 범위(range)-기반의 반복 루프
  - 타입 자동 추론 기능
  - 보편적인 초기화
  - 람다식



C++의 실행 속도



C++의 장점

- C++로 작성된 프로그램은 속도가 빠르다.
- C++은 멀티패러다임 프로그래밍을 지원한다. 즉 절차지향, 객체 지향, 제네릭 방법을 동시에 지원한다.
- 하드웨어에 접근할 수 있다.
- 메모리를 효율적으로 사용한다.
- C언어 프로그램을 그냥 가져다가 사용할 수 있다.
- 고성능의 게임이나 인공지능, 장치 드라이버에 적합하다.

객체 지향 프로그래밍

- '객체'라는 기본 단위로 나누고, 이 객체들의 상호작용으로 서술하는 방식이다.
- 객체 지향 프로그래밍에서는 데이터와 기능(연산)을 묶어 객체로 관리함.

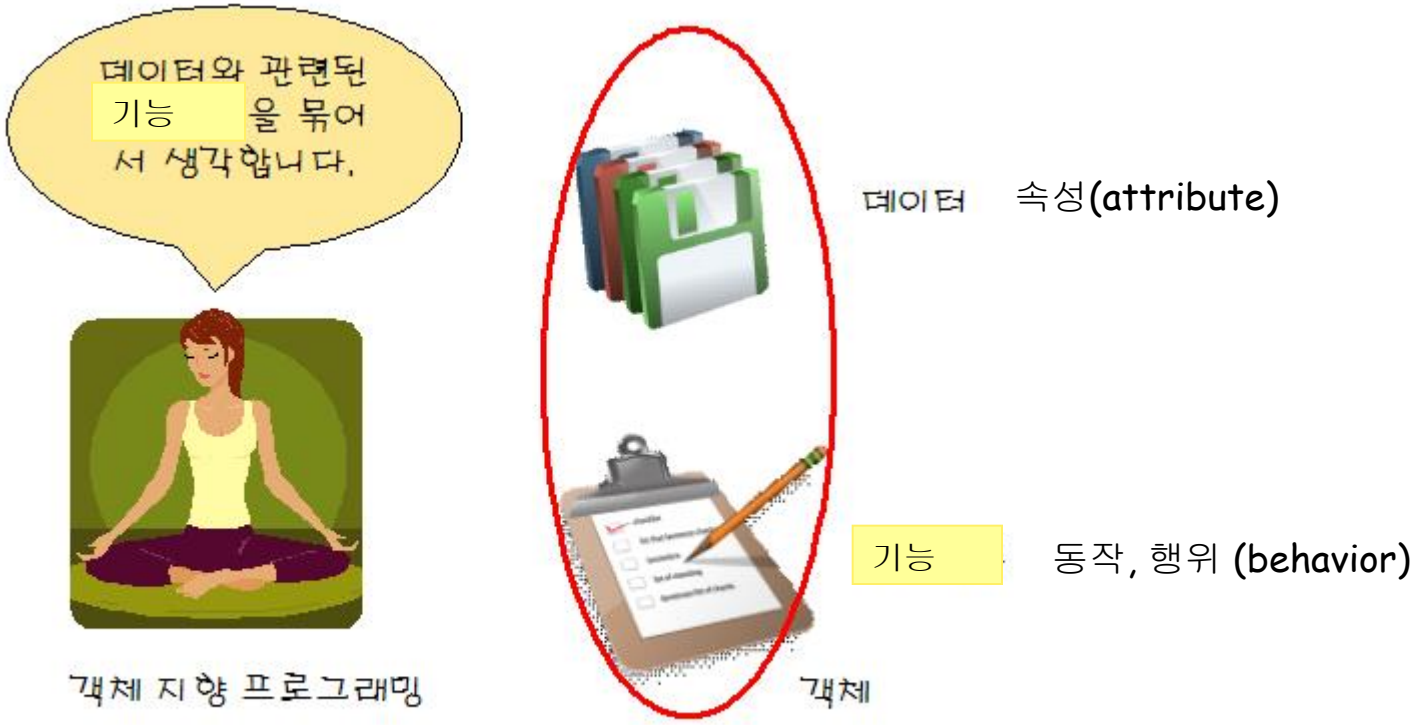
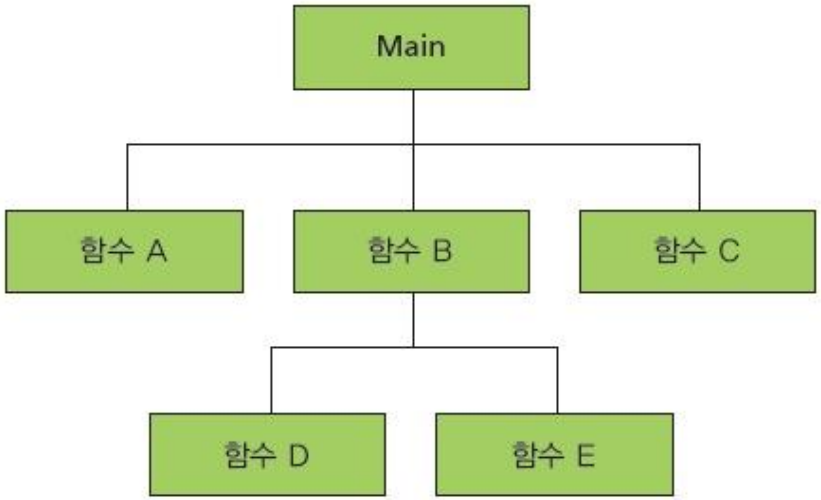
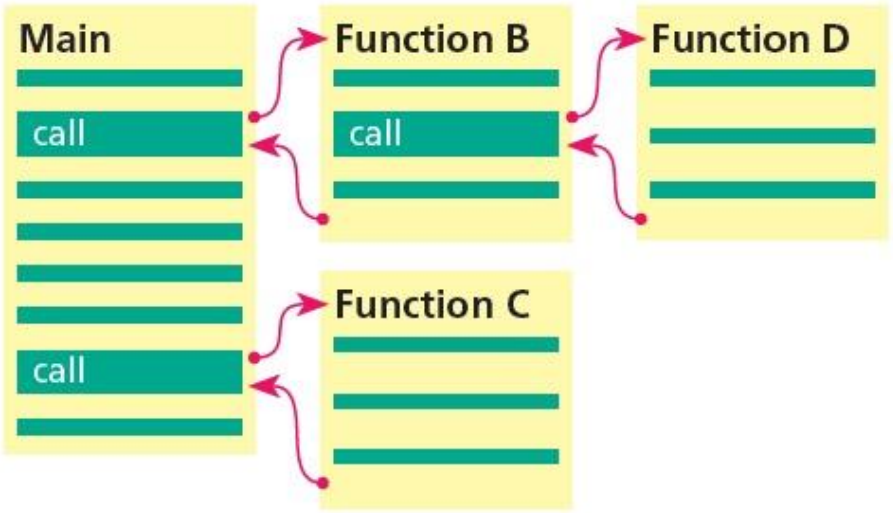


그림 1.9 객체 지향 프로그래밍

절차적 프로그래밍

- **절차 지향 프로그래밍(procedural programming)**은 기본적으로 프로시저를 사용하여 프로그램을 작성하는 프로그래밍 방식이다.





객체 지향 프로그래밍

- '객체'라는 기본 단위로 나누고, 이 객체들의 상호작용으로 서술하는 방식이다.
- 객체 지향 프로그래밍에서는 데이터와 기능(연산)을 묶어 객체로 관리함.

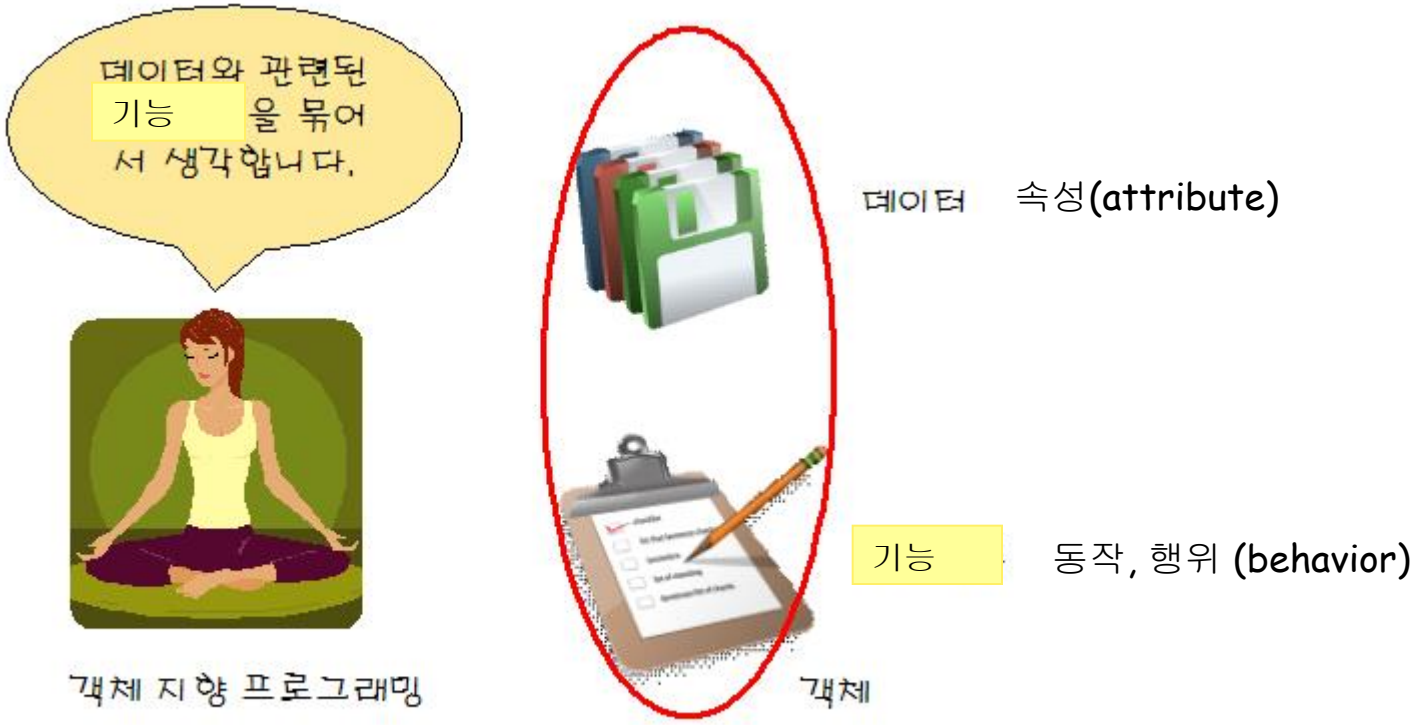


그림 1.9 객체 지향 프로그래밍

객체 지향의 개념들

- 캡슐화(encapsulation) : 데이터(속성)와 기능(행위)을 하나의 단위(클래스)로 묶는 것이다
- 정보 은닉(information-hiding) : 데이터에 대한 불필요한 접근을 차단하여서 데이터를 보호
- 상속(inheritance) : 비슷한 클래스가 이미 존재하고 있다면 그 클래스를 가져다가 사용하는 것
- 다형성(polymorphism) : 같은 이름의 함수나 연산자를 중복 정의하여서 상황에 따라서 가장 적절한 함수나 연산자를 프로그램이 자동적으로 선택

첫 번째 프로그램의 분석

test.cpp

```
// 첫번째 예제 프로그램

#include <iostream>
using namespace std;

int main(void)
{
        cout << "Hello World!" << endl;
        return 0;
}
```

주석

헤더파일

이름공간 설정

화면에 문자열 출력

실행결과

```
Hello World!
```

주석

- 주석(comment): 프로그램에 대한 설명

```
// 첫 번째 예제 프로그램
```

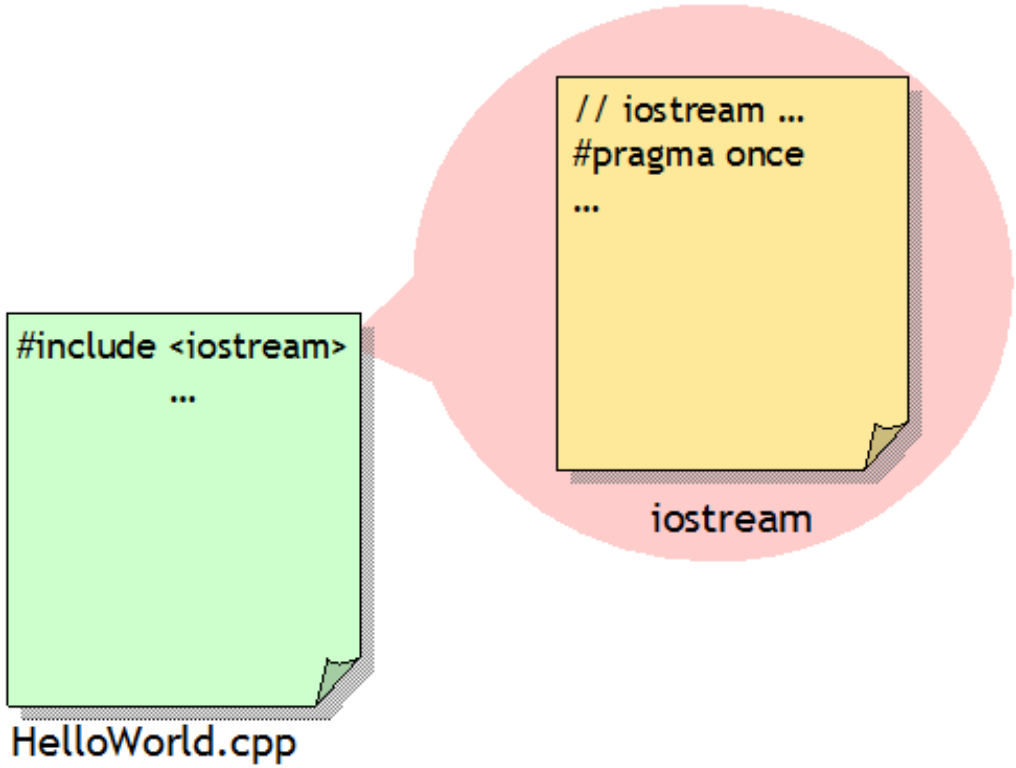
```
/* 한줄로된주석 */

int main()   /* 줄의일부분인주석 */

/* 여러
줄로
된주석 */
```

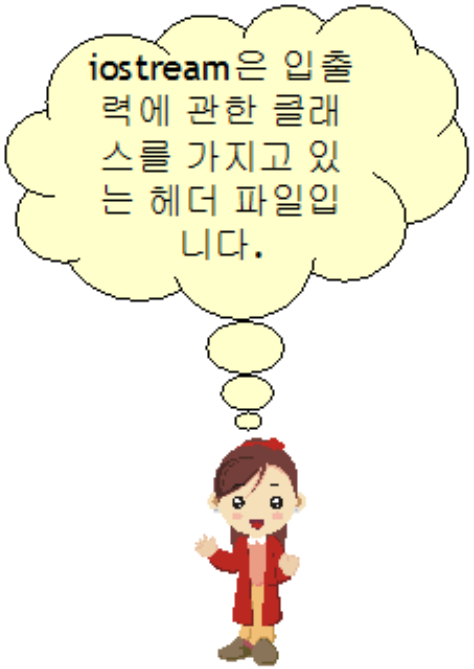
헤더 파일 포함

```
#include <iostream>
```

- #include는 소스 코드 안에 특정 파일을 현재의 위치에 포함
- 헤더 파일(header file): 컴파일러가 필요로 하는 정보를 가지고 있는 파일
- iostream: 표준 입출력 스트림
- 주의!: 전처리기 지시자 문장 끝에는 세미콜론을 붙이면 안 된다.

iostream 헤더 파일





이름 공간

```
using namespace std;
```

- **using**은 이름 공간을 지정하는 지시어이다.
- 변수 이름이나 함수 이름과 같은 수많은 이름(식별자)들은 이름 공간(**name space**)이라고 하는 영역으로 분리하여 저장
- 이름 공간 **std**를 사용한다는 의미

식별자를 사용하는 2가지 방법

1. std::cout << "Hello World!" << std::endl;

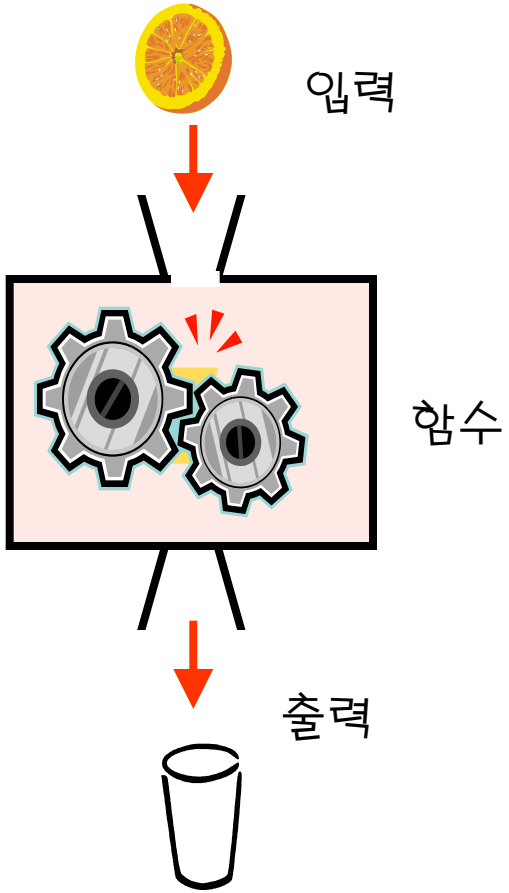
2. using namespace std;

cout << "Hello World!" << endl;

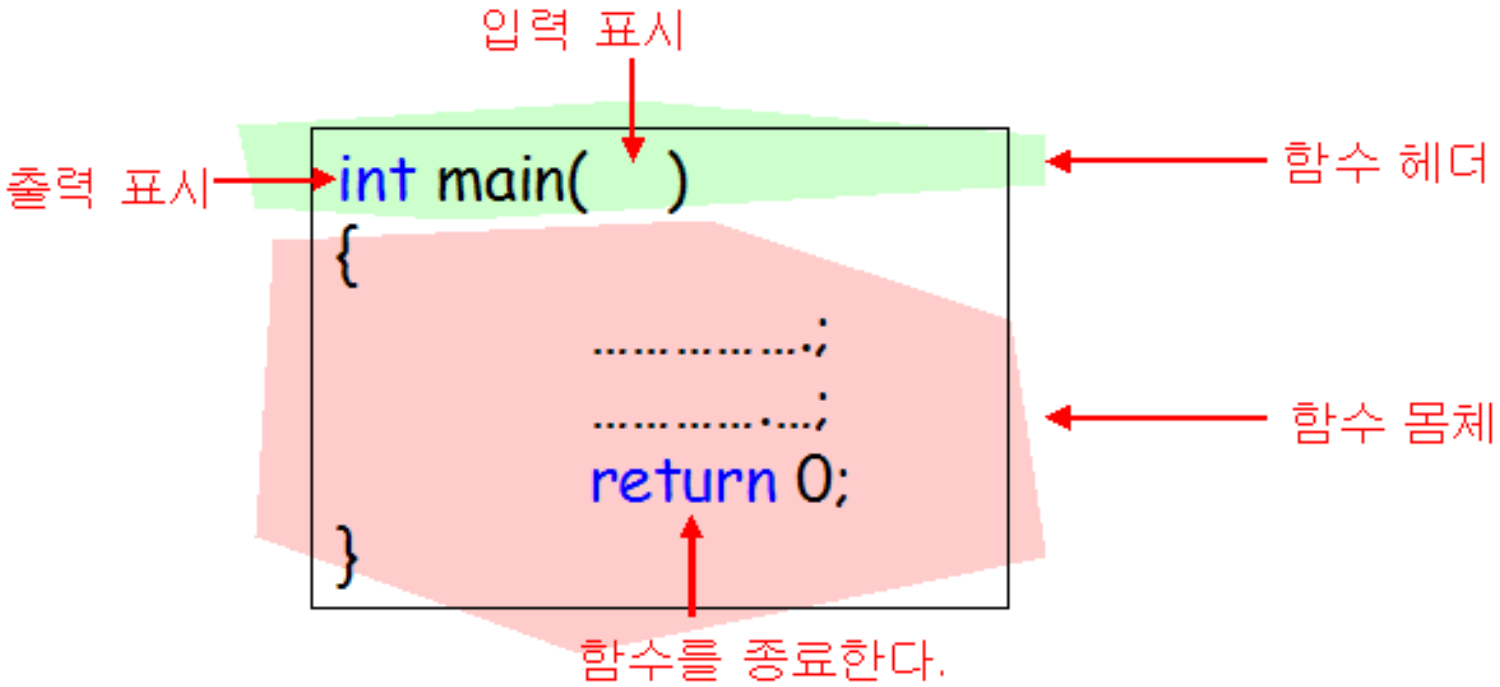
함수

```
int main()
```

- 함수(function): 특정한 작업을 수행하기 위하여 작성된 독립적인 코드
- main()은 가장 먼저 수행되는 함수

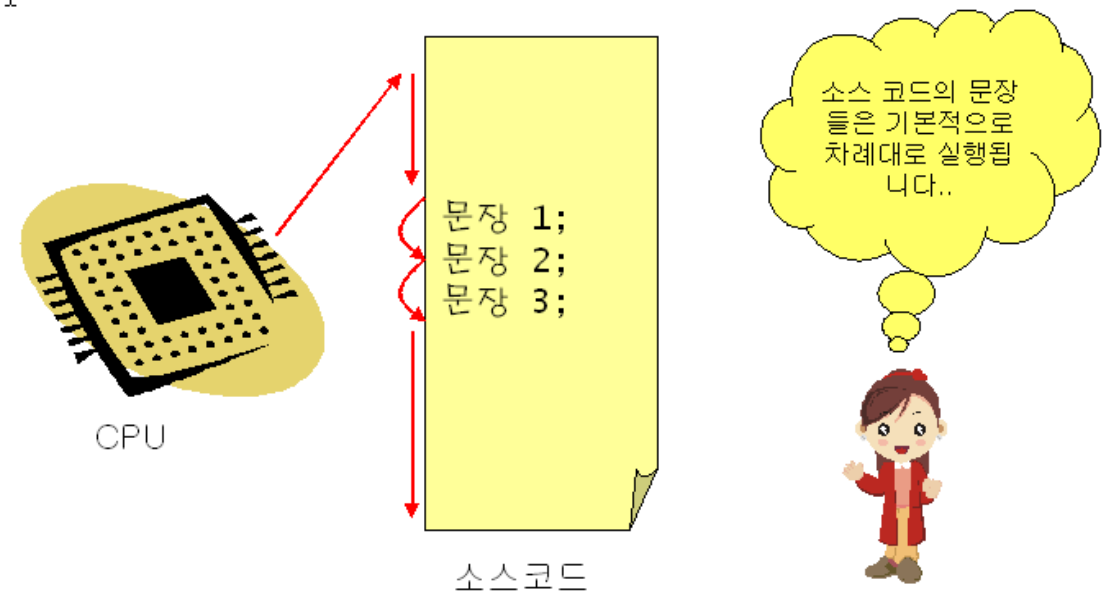


함수의 구조



문장

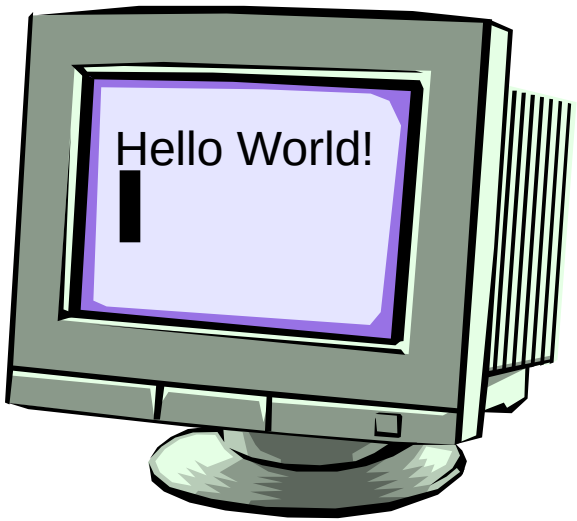
- 함수는 여러 개의 문장으로 이루어진다.
- 문장들은 순차적으로 실행된다.



출력 객체 cout

```
cout << “Hello World!” << endl;
```

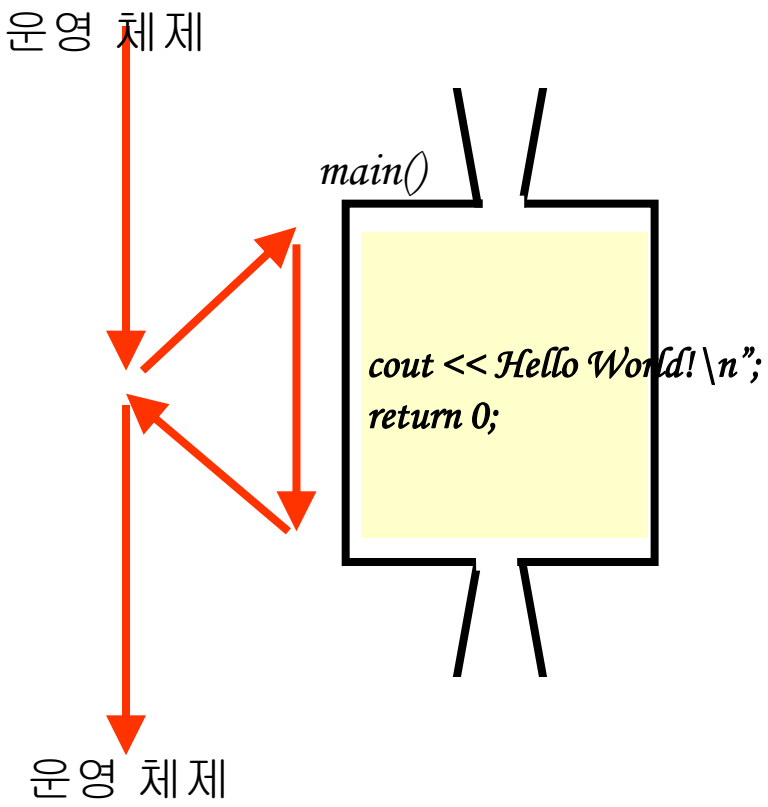
- cout은 컴파일러가 제공하는 객체로서 출력을 담당합니다.
- 큰따옴표 안의 문자열을 화면에 출력합니다.
- endl은 문장의 끝을 나타내는 기호(\n로 정의되어 있다).



함수 반환문

```
return 0;
```

- return은 함수의 결과값을 외부로 반환합니다.



첫 번째 버전

- 문장들은 순차적으로 실행된다는 사실 이용

```
#include <iostream>
using namespace std;

int main()
{
    cout <<" Hello World! ";
    cout << “Kim ChulSoo";
    return 0;
}
```

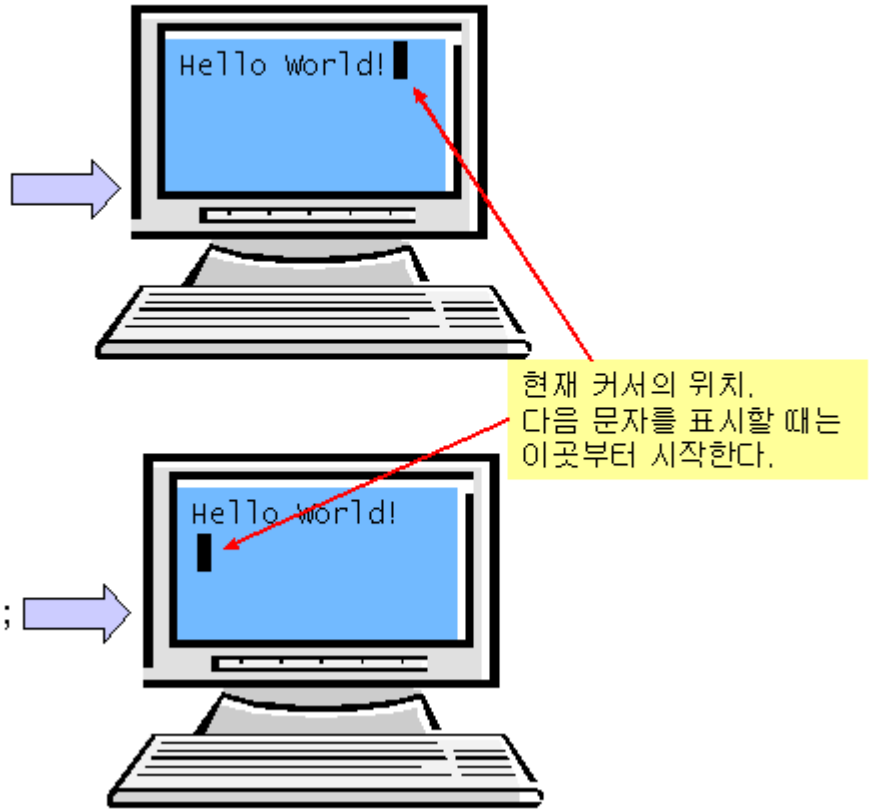
```
Hello World!Kim ChulSoo
```

우리가
원하는
결과가 아님!

줄바꿈 문자 \n

- 줄바꿈 문자인 \n은 화면에서 커서는 다음줄로 이동하게 한다.

cout << “Hello World!”;



cout << “Hello World!\n”;

변경된 프로그램

- 줄바꿈 문자를 포함하면 우리가 원하던 결과가 된다.

```
#include <iostream>
using namespace std;

int main()
{
    cout <<“Hello World!\n";
    cout << “Kim ChulSoo\n”;
    return 0;
}
```

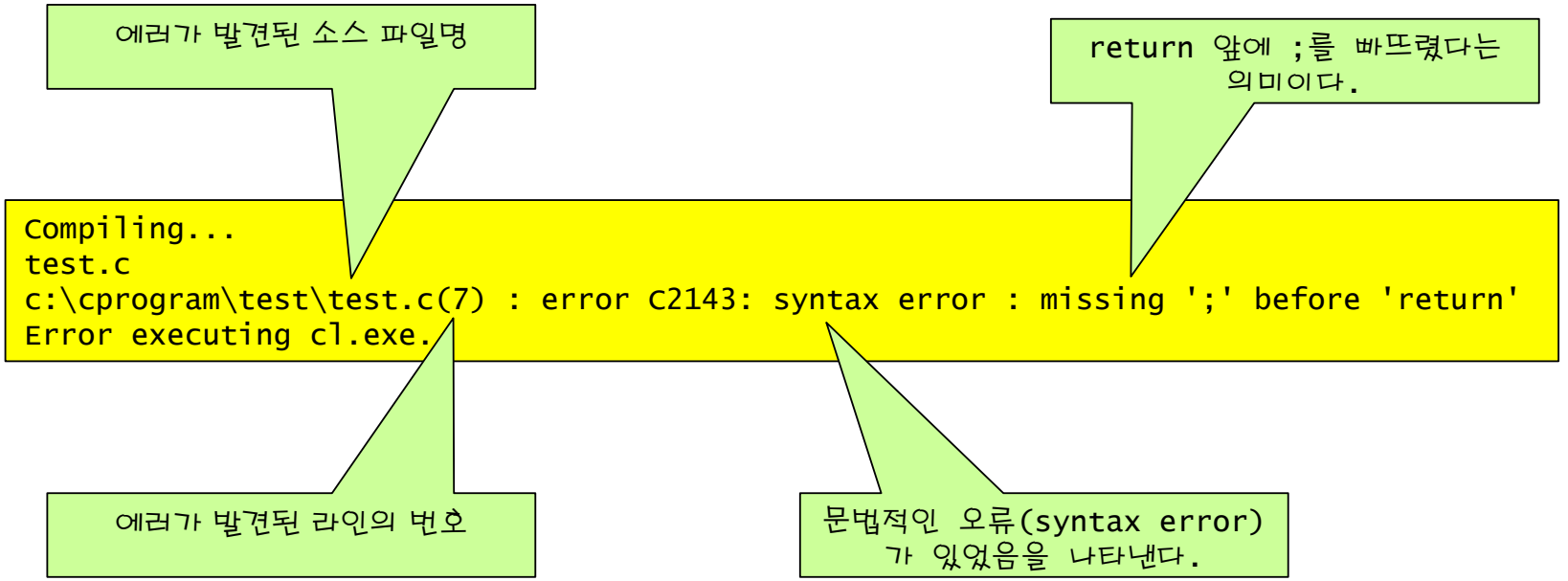
```
Hello World!
Kim ChulSoo
```

응용 프로그램 #2

- 역시 문장들은 순차적으로 수행된다는 점을 이용한다.

```
#include <iostream>
using namespace std;

int main()
{
    cout <<“3X1=3\n";
    cout <<“3X2=6\n";
    cout <<“3X3=9\n";
    return 0;
}
```

오류 메시지의 분석



변수

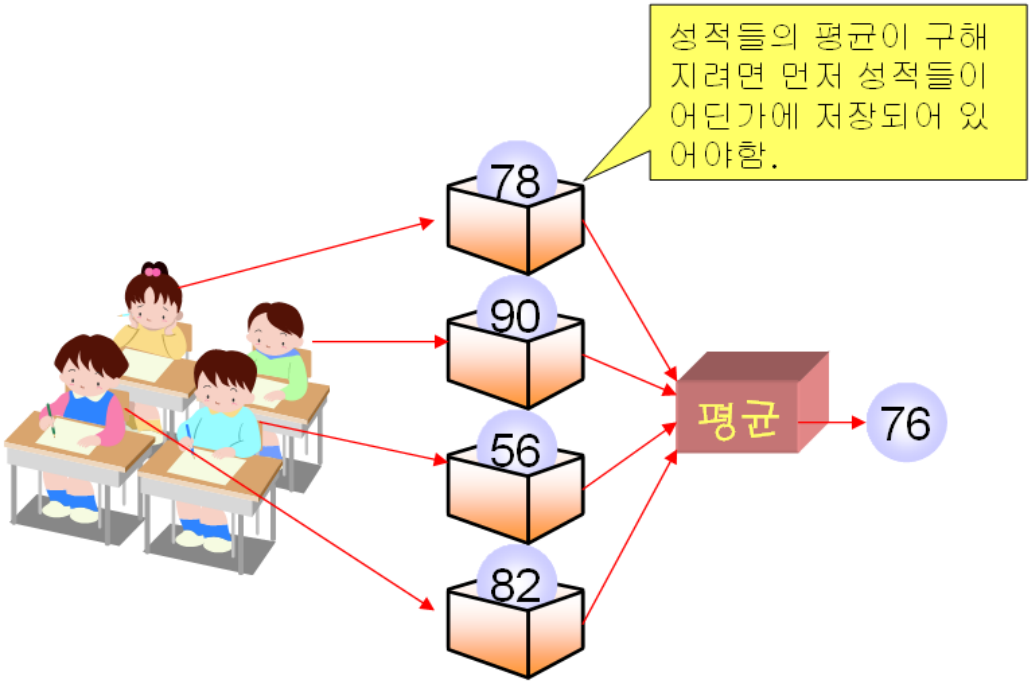
Q) 변수(variable)이란 무엇인가?

A) 프로그램에서 데이터를 저장하는 공간

Q) 변수는 왜 필요한가?

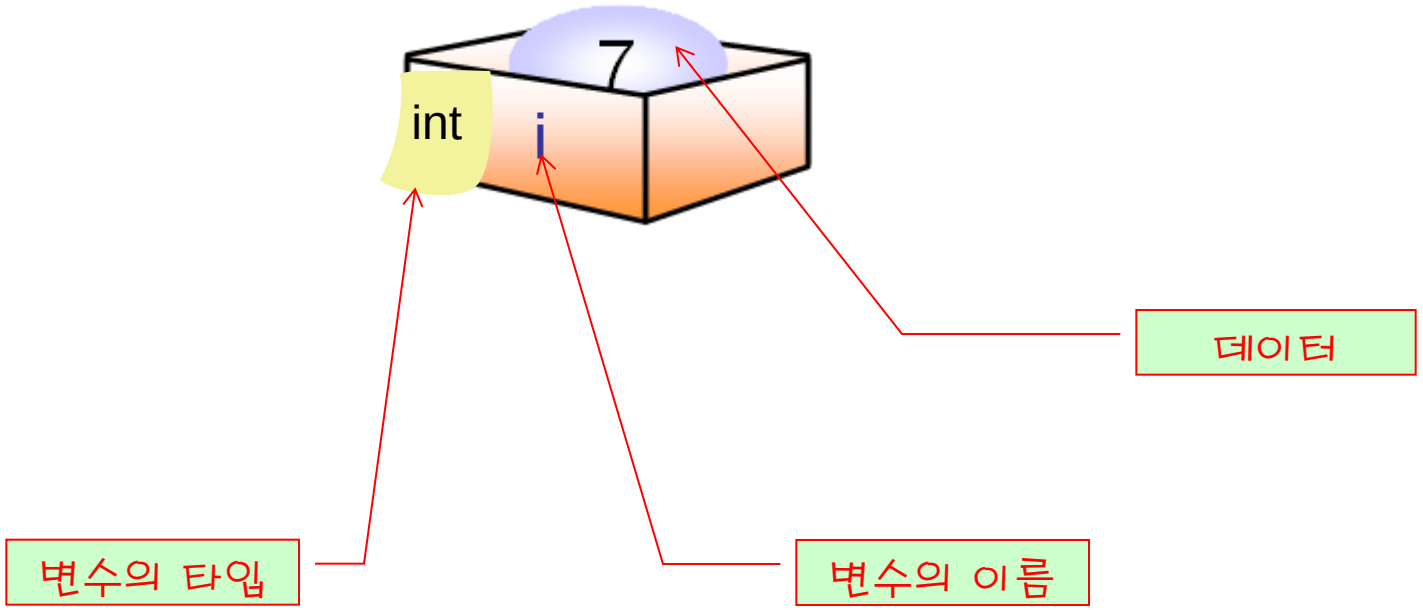
A) 데이터가 입력되면 어딘가에 저장해야만 다음에 사용할 수 있다

.



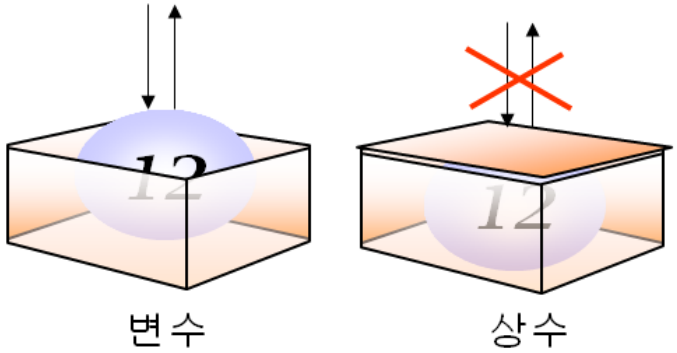
변수 = 상자

- 변수는 물건을 저장하는 상자와 같다.



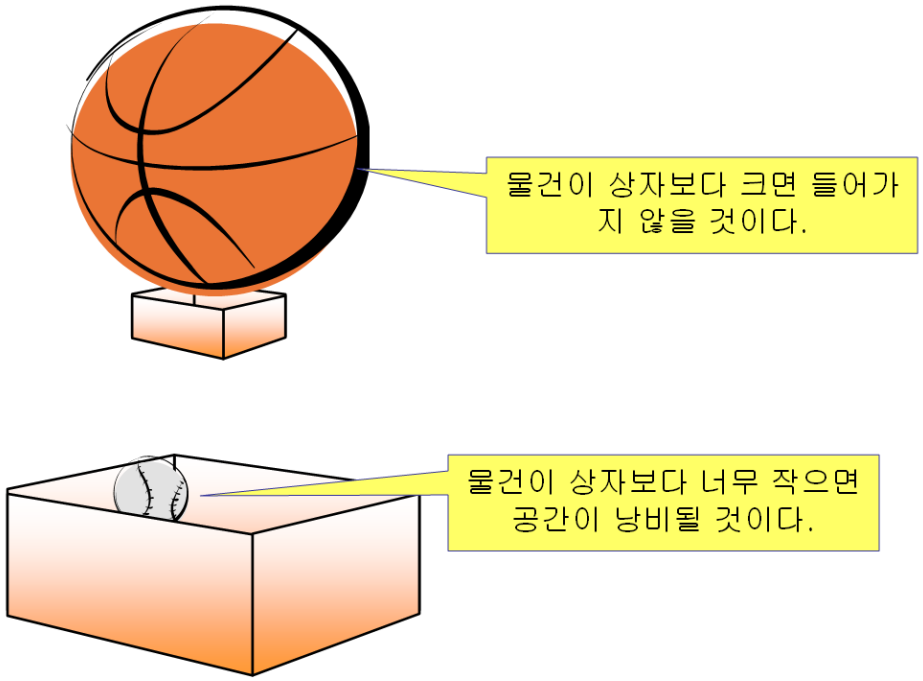
변수와 상수

- 변수(variable): 저장된 값의 변경이 가능한 공간
- 상수(constant): 저장된 값의 변경이 불가능한 공간
  (예) 3.14, 100, ‘A’, “Hello World!”



자료형

- 자료형(data type): 데이터의 타입(종류)
  - (예) 정수형 , 실수형



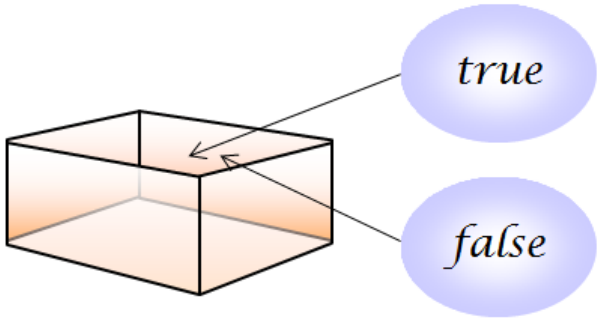
자료형의 종류

자료형			설명	바이트수	범위
정수형	부호있음	short	short형 정수	2	-32768 ~ 32767
		int	정수	4	-2147483648 ~ 2147483647
		long	long형 정수	4	-2147483648 ~ 2147483647
		long long	long long형 정수	8	-2^63 ~ 2^63 - 1
	부호없음	unsigned short	부호없는 short형 정수	2	0 ~ 65535
		unsigned int	부호없는 정수	4	0 ~ 4294967295
		unsigned long	부호없는 long형 정수	4	0 ~ 4294967295
		unsigned long long	부호없는 long long형 정수	8	0 ~ 2^64 - 1
문자형	부호있음	char	문자 및 정수	1	-128 ~ 127
	부호없음	unsigned char	문자 및 부호없는 정수	1	0 ~ 255
부동소수점형		float	단일정밀도 부동소수점	4	1.2E-38 ~ 3.4E38
		double	두배정밀도 부동소수점	8	2.2E-308 ~ 1.8E308
부울형		bool	참이나 거짓을 나타낸다.	1	true, false

Bool형

- 논리형의 변수는 참 또는 거짓의 값만을 가질 수 있다.

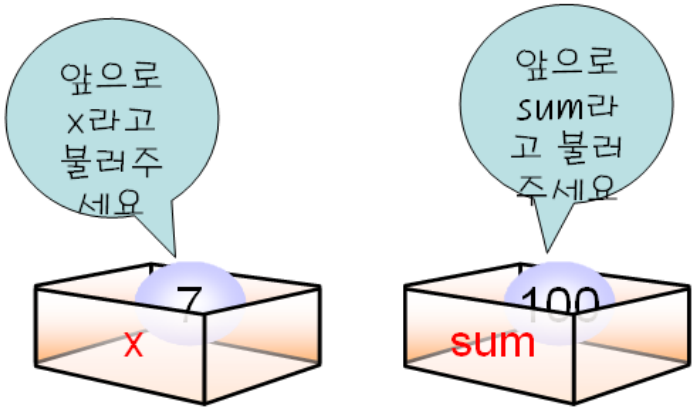
  bool condition = true;
- and(&&), or(||), not(!) 연산자



bool형 변수

변수의 이름짓기

- 식별자(identifier): 식별할 수 있게 해주는 이름
  - 변수 이름
  - 함수 이름



식별자를 만드는 규칙

- 알파벳 문자와 숫자, 밑줄 문자 _로 구성
- 첫 번째 문자는 반드시 알파벳 또는 밑줄 문자 _
- 대문자와 소문자를 구별
- C 언어의 키워드와 똑같은 이름은 허용되지 않는다.

(Q) 다음은 유효한 식별자인가?

sum O
_count O
king3 O
n_pictures O
2nd_try X // 숫자로 시작
Dollor# X // #기호
double X // 키워드

키워드

- 키워드(keyword): C++언어에서 고유한 의미를 가지고 있는 특별한 단어
- 예약어(reserved words) 라고도 한다.

C언어의 키워드

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

C++언어의 키워드

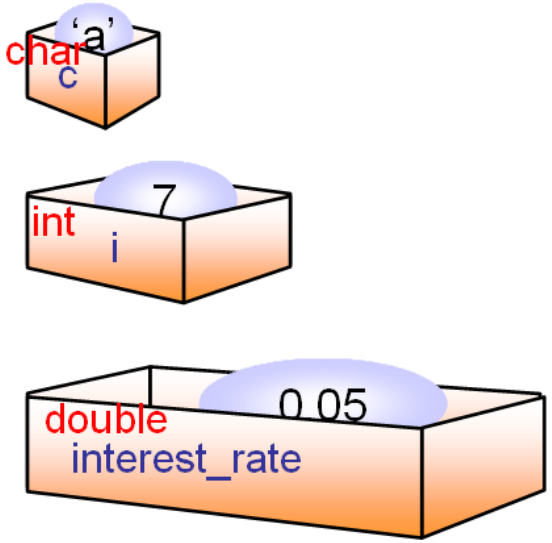
asm	false	protected	try
bool	friend	public	typeid
catch	inline	reinterpret_cast	typename
class	mutable	static_cast	using
const_cast	namespace	template	virtual
delete	new	this	wchar_t
dynamic_cast	operator	throw	
explicit	private	true	

변수 선언

- 변수 선언: 컴파일러에게 어떤 변수를 사용하겠다고 미리 알리는 것

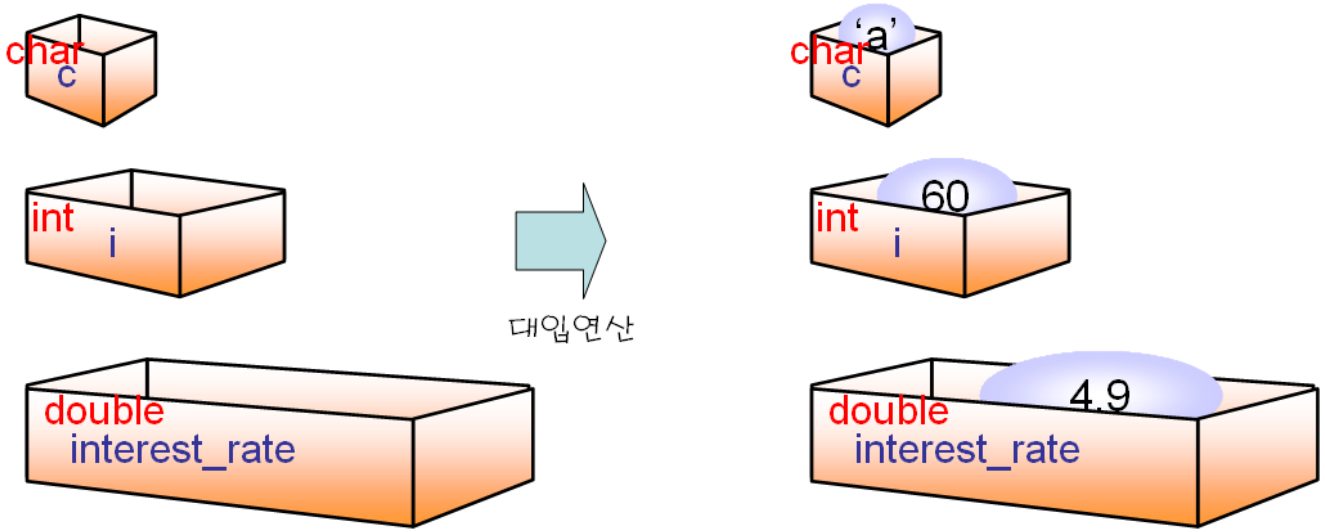
```
자료형    변수이름;
```

- 변수 선언의 예
  - char c;
  - int i;
  - double interest_rate;
  - int height, width;



변수에 값을 저장하는 방법

```
char c;                          // 문자형 변수 c 선언
int i;                   // 정수형 변수 i 선언
double interest_rate;            // 실수형 변수 interest_rate 선언


c = 'a';                 // 문자형 변수 c에 문자 'a'를 대입
i = 60;                          // 정수형 변수 i에 60을 대입
interest_rate = 4.9;     // 실수형 변수 interest_rate에 82.9를 대입
```

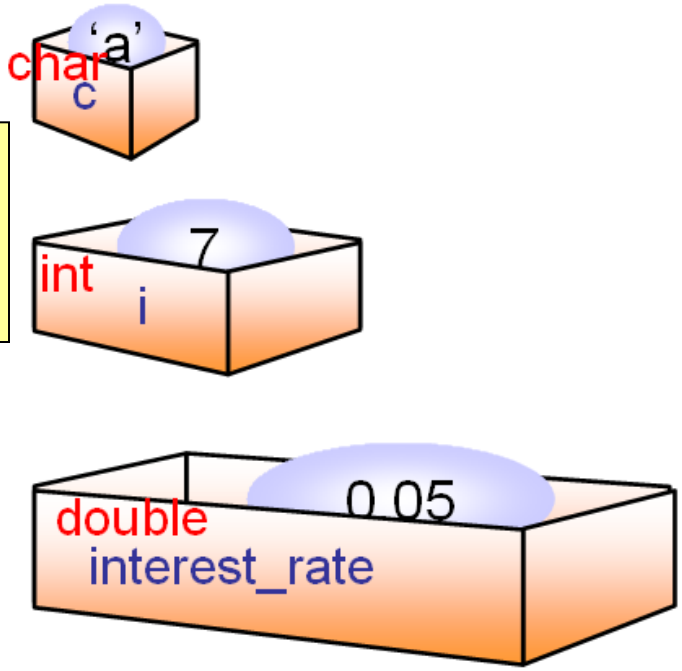


변수의 초기화

```
자료형    변수이름 = 초기값;
```

- 변수 초기화의 예

```
char c = 'a';
int i = 7;
double interest_rate = 0.05;
```



문자열 타입

- string 타입을 제공한다.

string.cpp

```
#include <iostream>
#include <string>
using namespace std;

int main(void)
{
   string s1 = "Good";
   string s2 = "Morning";
   string s3 = s1 + " " + s2 + "!\n";
   cout << s3;
   return 0;
}
```

이 헤더파일을 반드시 포함하여야 한다.

실행결과

```
Good Morning!
```

문자열 타입

- 문자열의 비교는 == 연산자로 가능

```
int main(void)
{
   string s1 = "Good";
   if( s1 == "Good" ){
         ....
   }
   else if( s1 == "Bad" ){
         ....
   }
   return 0;
}
```

기호 상수

- 기호 상수(symbolic constant): 기호를 이용하여 상수를 표현한 것
- (예)
  - area = 3.141592 * radius * radius;
  - area = PI * radius * radius;

  - income = salary - 0.15 * salary;
  - income = salary - TAX_RATE * salary;
- 기호 상수의 장점
  - 가독성이 높아진다.
  - 값을 쉽게 변경할 수 있다.

기호 상수의 장점

상수가 사용된 모든 곳을 변경하여야 한다.

```
income = salary-0.15*salary;
  ...
expenditure += 0.15*salary;
```

기호상수의 정의만 바꾸면 된다.

```
#define TAX_RATE 0.15

income = salary-TAX_RATE*salary;
  ...
expenditure += TAX_RATE*salary;
```

기호 상수를 만드는 방법

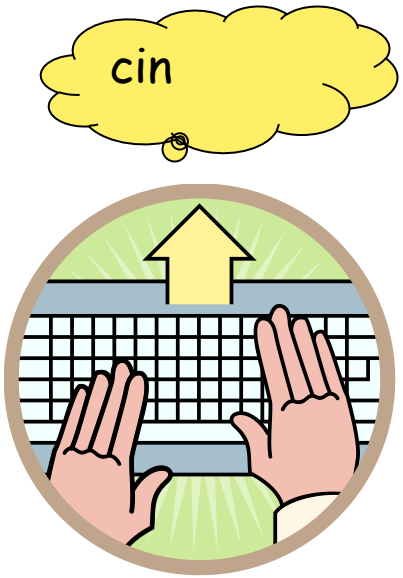
const 키워드 이용

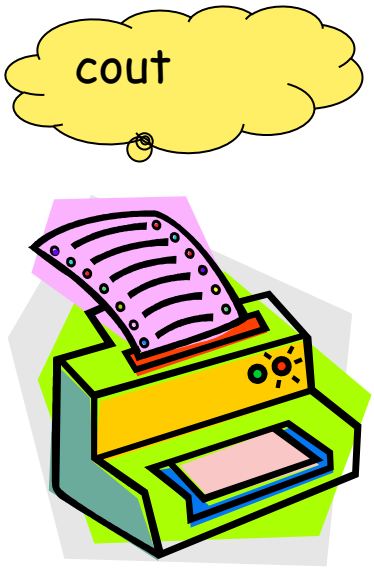
```
#include <iostream>
using namespace std;        // 이름공간설정
int main()
{
    const int MONTHS = 12;// 기호상수선언
    double m_salary, y_salary;        // 변수선언

    cout << "월급을입력하시요: ";    // 입력안내문
    cin >> m_salary;

    y_salary = 12 * m_salary;         // 순수입계산
    cout << "연봉은" << y_salary << "입니다" << endl;
    return 0;
}
```

기호 상수 정의

출력과 입력

- C++에서는 콘솔 입력은 cin 객체가, 콘솔 출력은 cout 객체가 담당
- 이들은 모두 iostream 라이브러리에 포함





출력

형식	cout << 값
설명	값을 텍스트로 변환하여 표준 출력 스트림에 출력한다.
예	cout << "Hello World!";

```
cout << 100;

int i = 100;
cout << i;

cout << "변수 i의 값은 " << i << "입니다."
```

출력

- cout은 스스로 변수의 자료형에 따라 적절하게 출력할 수 있다.

```
int i;
float f;


cout << i;  // 정수 형식으로 i의 값이 출력된다.
cout << f; // 실수 형식으로 f의 값이 출력된다.
```

입력

형식	cin >> 변수
설명	키보드에서 데이터를 읽어서 변수에 저장한다.
예	cin >> value;

```
int main(void)
{
        int i;
        cin >> i;   // 정수를 읽어서 i에 저장
        double f;
        cin >> f;   // 실수를 읽어서 f에 저장
        return 0;
}
```

예 제

cin.cpp

```
#include <iostream>
#include <string>
using namespace std;

int main(void)
{
   string name;
   cout << "이름을 입력하시오: ";
   cin >> name;
   cout << name << "을 환영합니다." << endl;
   return 0;
}
```

문자열이 name으로 입력된다.

실행결과

```
이름을 입력하시오: 홍길동
홍길동을 환영합니다.
```

예 제

```
#include <iostream>
using namespace std;

int main()
{
    int x=0, y=0, z=0;

    cout << (2 > 3 || 6 > 7) << endl;
    cout << (2 || 3 && 3 > 2) << endl;
    cout << (x = y = z = 1) << endl;
    cout << (- ++x + y--) << endl;

    return 0;
}
```

```
0
1
1
-1
```

auto 키워드

- 자동 타입 추론(automatic type deduction)

```
auto d = 1.0;
```

```
auto add(int x, int y)
{
        return x + y;
}

int main()
{
        auto sum = add(5, 6); 된다.
        return 0;
}
```