



# 프로그래밍의 개념과 이해

2ND  
EDITION





# 이번 장에서 학습할 내용



- 프로그래밍 개념
- 프로그래밍 언어
- 알고리즘
- 프로그램 개발 과정

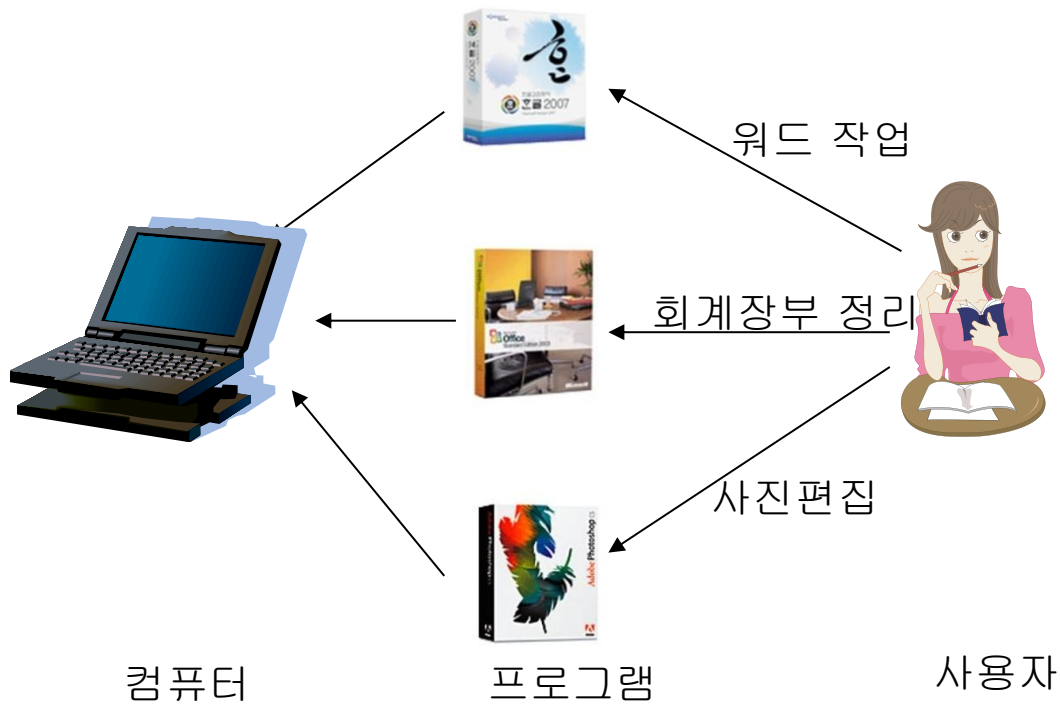
프로그램을  
작성하기에 앞서서  
중요한 개념들을  
살펴봅니다..





# 프로그램이란?

- 컴퓨터 = 하드웨어 + 소프트웨어(프로그램)
- 컴퓨터를 범용적으로 만드는 것은 바로 프로그램

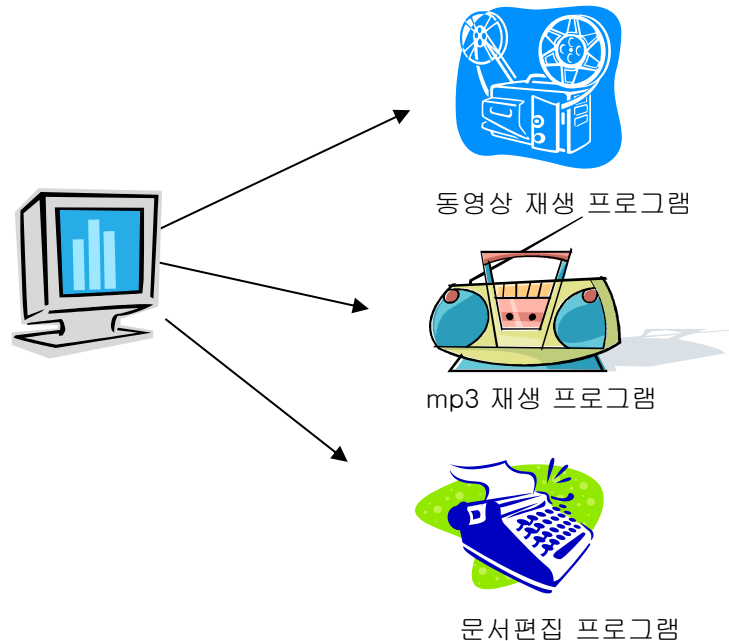




# 프로그램

Q) 왜 컴퓨터에서는 가전제품처럼 프로그램 설치 없이 바로 동작되도록 하지 않고 불편하게 사용자가 프로그램을 설치하게 하였을까 ?

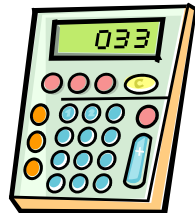
A) 컴퓨터를 범용적인 기계로 만들기 위해서이다.  
컴퓨터는 프로그램만 바꾸어주면 다양한 작업을 할 수 있다.





# 계산기와 컴퓨터의 차이

계산기는 정해진  
기능만을  
수행한다.  
기능을 변경할  
수 없다.



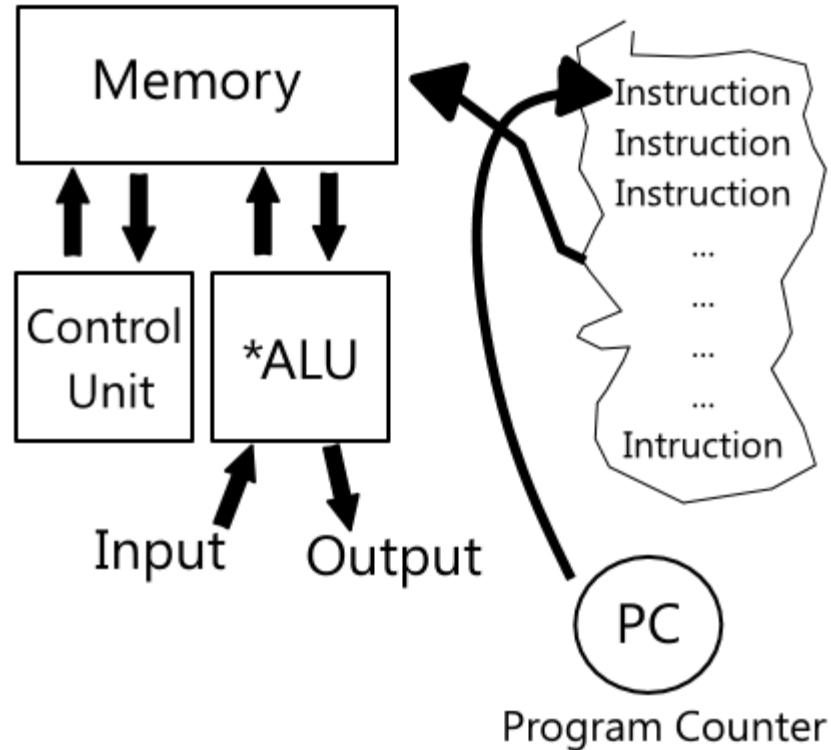
프로그램이라는  
개념을 도입하여  
수행하는 기능을  
쉽게 변경할 수  
있다.





# 컴퓨터의 정의

- 컴퓨터(computer)는 단순히 계산(compute)만 하는 기계가 아니다.
- 현대적인 의미에서의 컴퓨터는 프로그램(명령어들의 리스트)에 따라 데이터를 처리하는 기계라고 할 수 있다

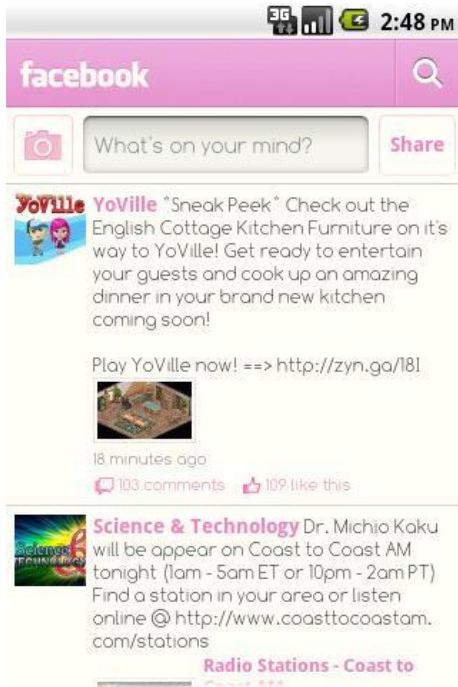


\*ALU(Arithmetic Logic Unit)



# 스마트폰도 컴퓨터의 일종

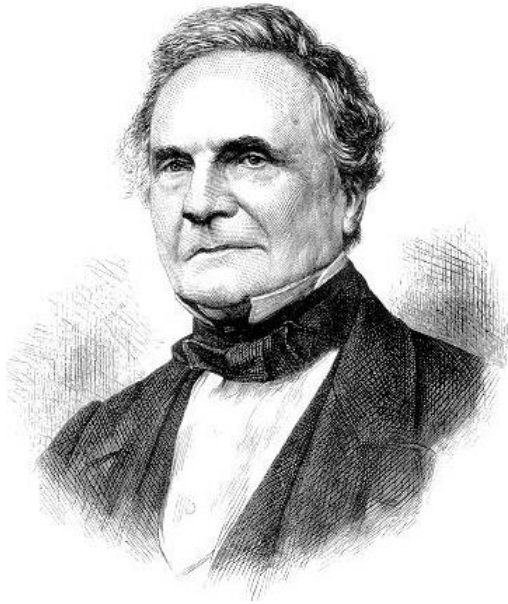
- **피쳐폰:** 미리 설정된 기능만 가능
- **스마트폰:** 애플리케이션만 변경하면 다양한 용도로 사용가능





# 프로그램의 역사

- 프로그래밍이 가능한 최초의 기계: 해석 기관(**Analytical Engine**)
- 만드인: 찰스 배비지
- 수천 개의 기어, 바퀴, 축, 레버 등이 증기로 작동







# 최초의 프로그래머

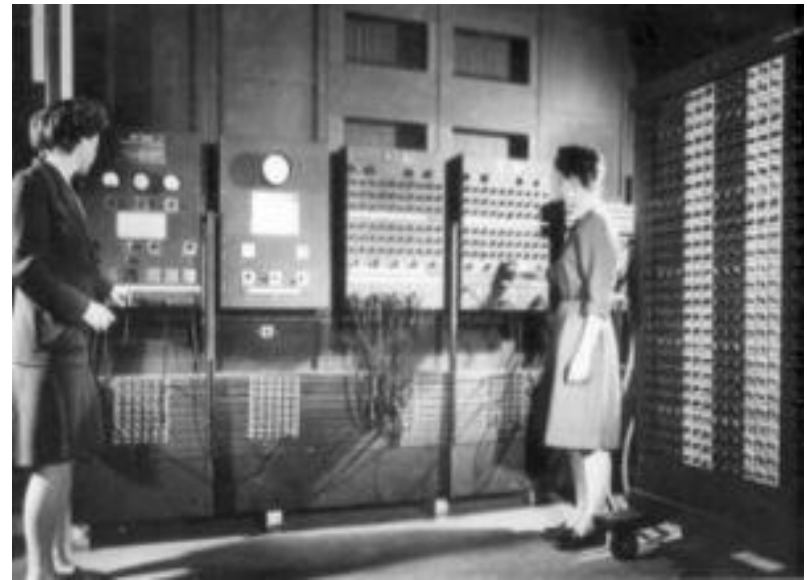
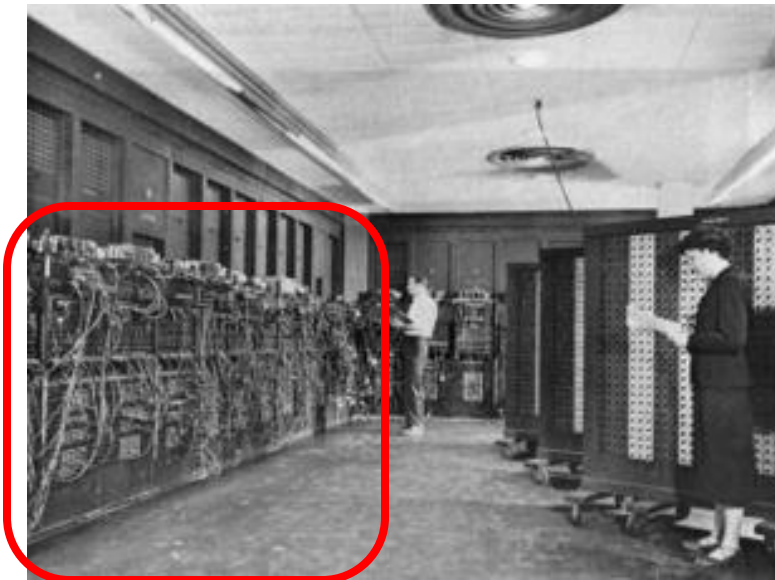
- 프로그램을 최초로 만든 사람은 **에이다 러브레이스(Ada Lovelace)**
- 에이다는 대문호 바이런의 친딸
- 배비지의 해석 기관에 매료되어 해석 기관을 위한 프로그램을 개발하였다.
- 서브루틴(subroutine), 루프(loop), 점프(jump) 등의 핵심적인 컴퓨터 프로그래밍 기본 원리를 고안





# 초기 컴퓨터의 프로그래밍

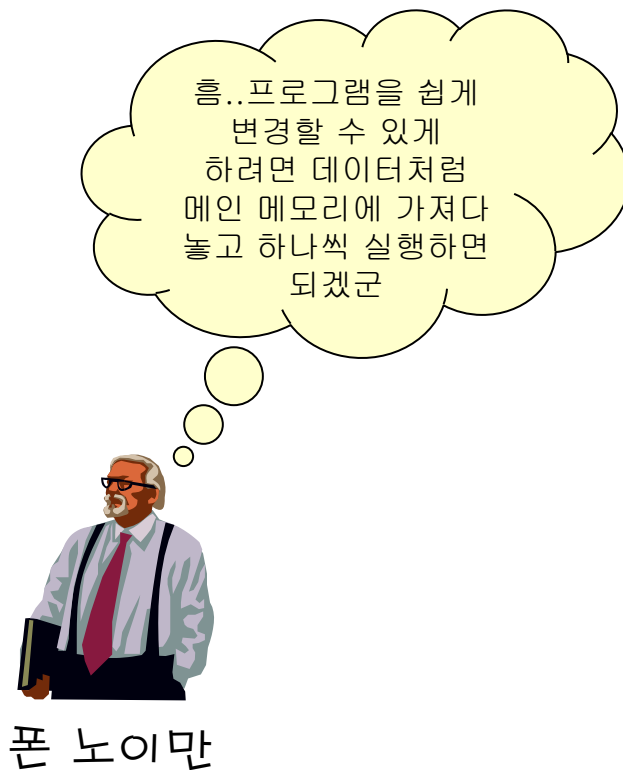
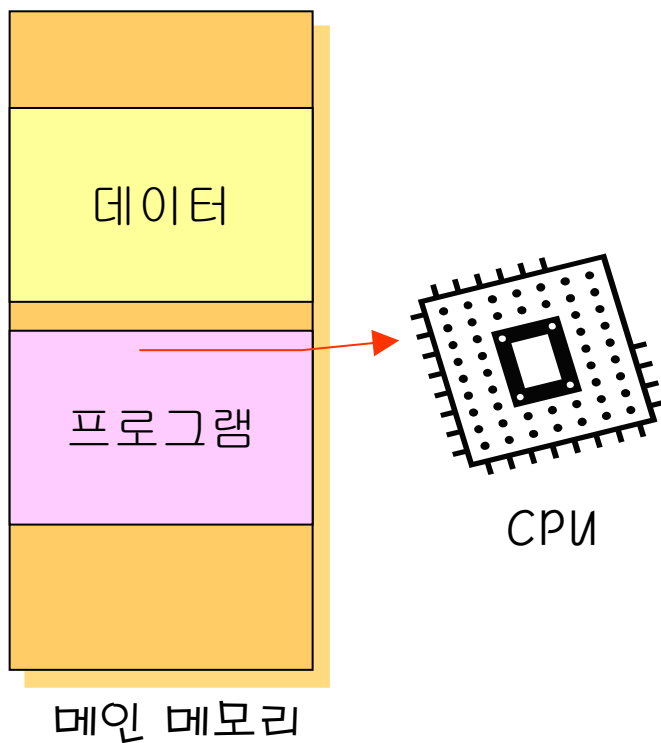
- 초기 컴퓨터인 **ENIAC**의 프로그램은 스위치에 의하여 기억되었고 프로그램을 변경할 때마다 그 많은 스위치들을 처음부터 다시 연결하여야 했다.





# 폰노이만 구조

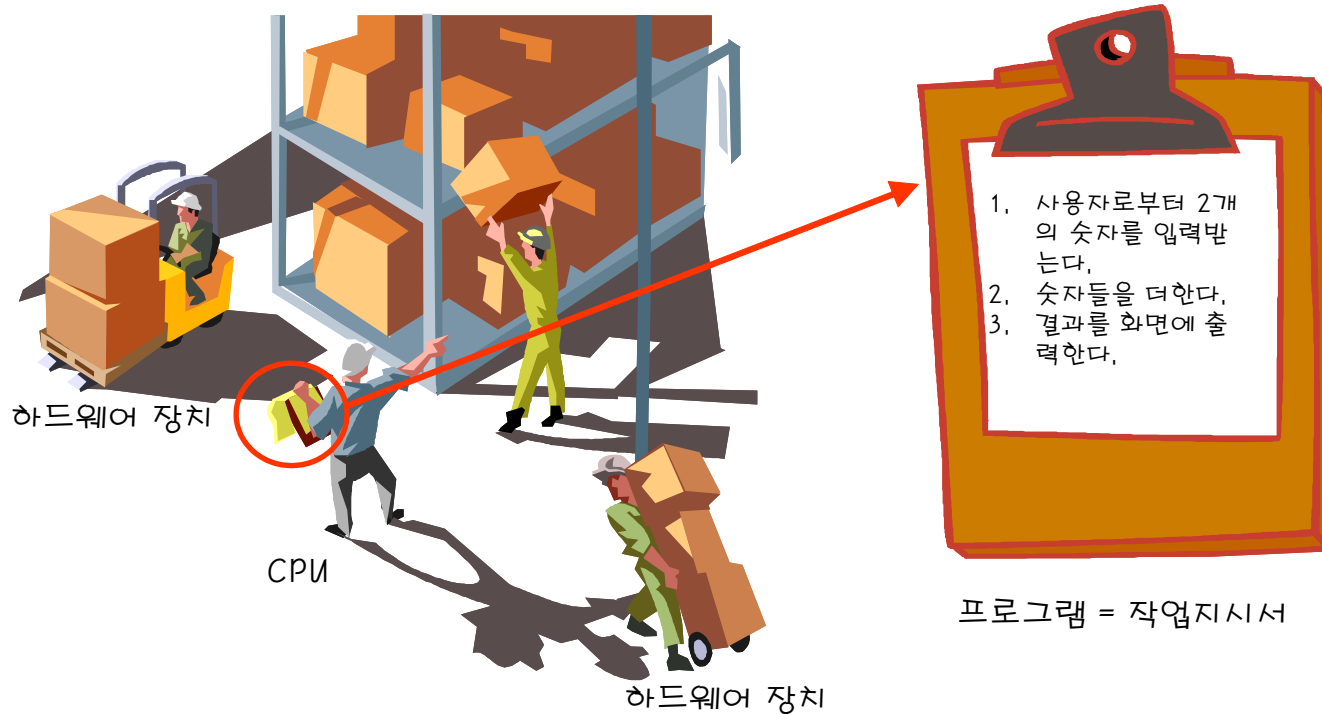
- 프로그램은 **메인 메모리에 저장**된다.
- 메인 메모리에 저장된 프로그램에서 명령어들을 순차적으로 가져와서 실행한다.





# 프로그램 == 작업지시서

- 프로그램: 컴퓨터에게 해야 할 작업의 내용을 알려주는 문서





# 이번 장에서 학습할 내용



- 프로그래밍 개념
- 프로그래밍 언어
- 알고리즘
- 프로그램 개발 과정

프로그램을  
작성하기에 앞서서  
중요한 개념들을  
살펴봅니다..

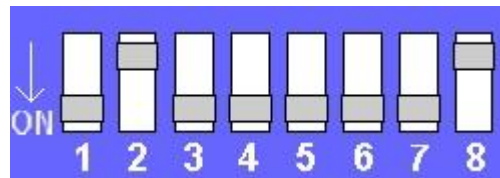




# 기계어

Q) 컴퓨터가 이해할 수 있는 언어는 어떤 것인가?

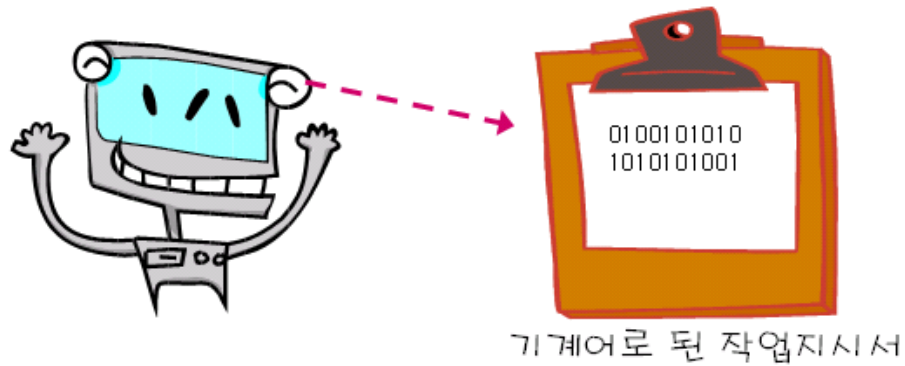
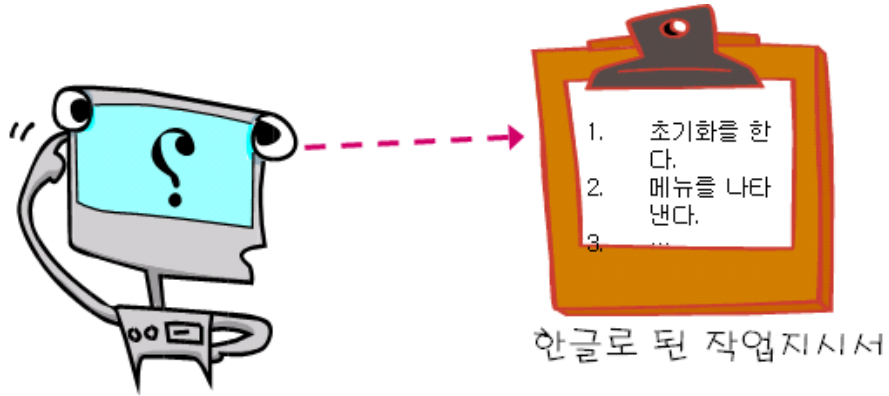
- A) 컴퓨터가 알아듣는 언어는 한가지이다. 즉 0과 1로 구성되어 있는 “001101110001010...”과 같은 기계어이다.
- A) 컴퓨터는 모든 것을 0과 1로 표현하고 0과 1에 의하여 내부 스위치 회로들이 ON/OFF 상태로 변경되면서 작업을 한다.





# 기계어

- 컴퓨터는 기계어를 바로 이해할 수 있다.





# 기계어

- 기계어의 예

Address:		0x401058															
00401058	8B 45 FC 03 45 F8 89 45 F4	8B F4 6A 64															
00401073	A3 42 00 3B F4 E8 B3 06 00 00	8B F4 6A															
0040108E	CC A3 42 00 3B F4 E8 97 06 00 00	8B 45															
004010A9	8B 55 08 52 E8 58 FF FF FF	83 C4 08 85															

26:		
27:	int a, b, c;	
28:		
29:	c = a + b;	
00401058	mov	eax,dword ptr [ebp-4]
0040105B	add	eax,dword ptr [ebp-8]
0040105E	mov	dword ptr [ebp-0Ch],eax
30:		





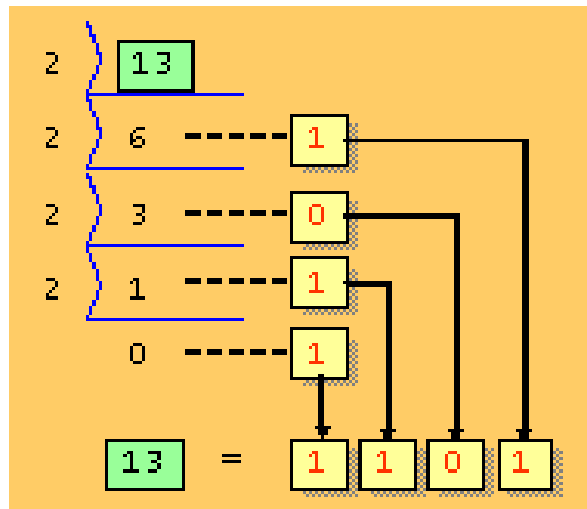
# 이진수

Q) 이진수는 십진수와 무엇이 다른가?

A) 이진수는 0과 1로만 구성되어 있다.

Q) 십진수를 이진수로 바꾸려면?

A) 십진수를 이진수로 바꾸려면 십진수를 2로 나누고 나머지를 기록하는 작업을 몫이 0이 될 때까지 되풀이하면 된다.

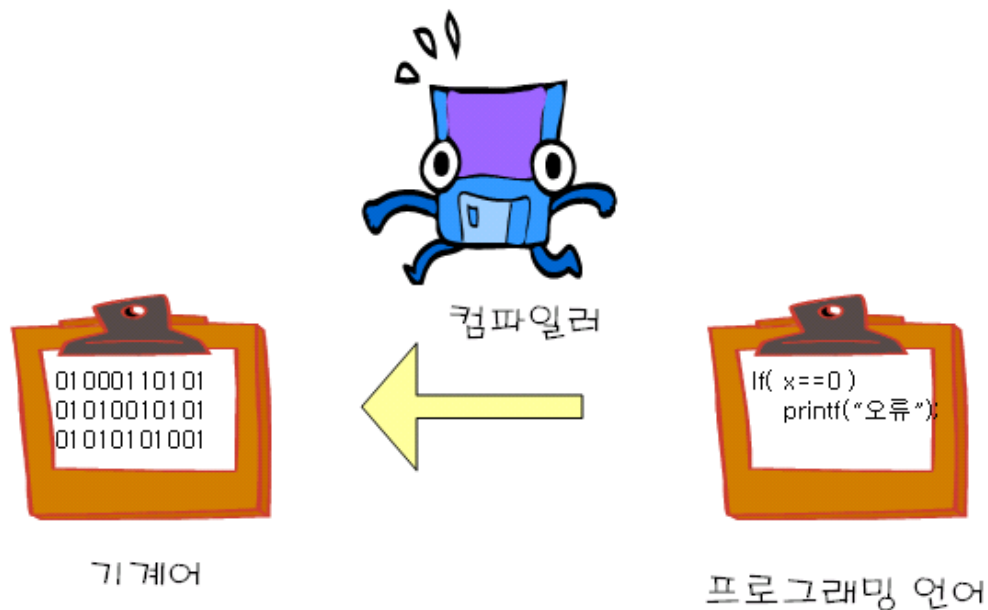




# 프로그래밍 언어의 필요성

Q) 그렇다면 인간이 기계어를 사용하면 어떤가?

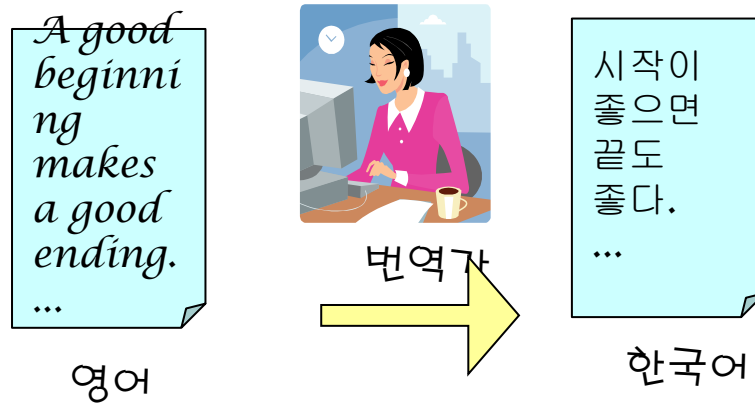
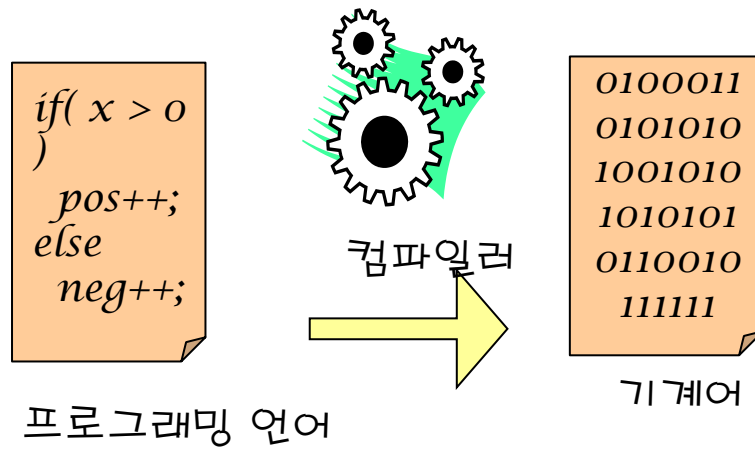
- 기계어를 사용할 수는 있으나 이진수로 프로그램을 작성하여야 하기 때문에 아주 불편하다.
- 프로그래밍 언어는 자연어와 기계어 중간쯤에 위치
- 컴파일러가 프로그래밍 언어를 기계어로 통역





# 컴파일러

- 컴파일러(compiler)는 인간과 컴퓨터 사이의 통역이라 할 수 있다.



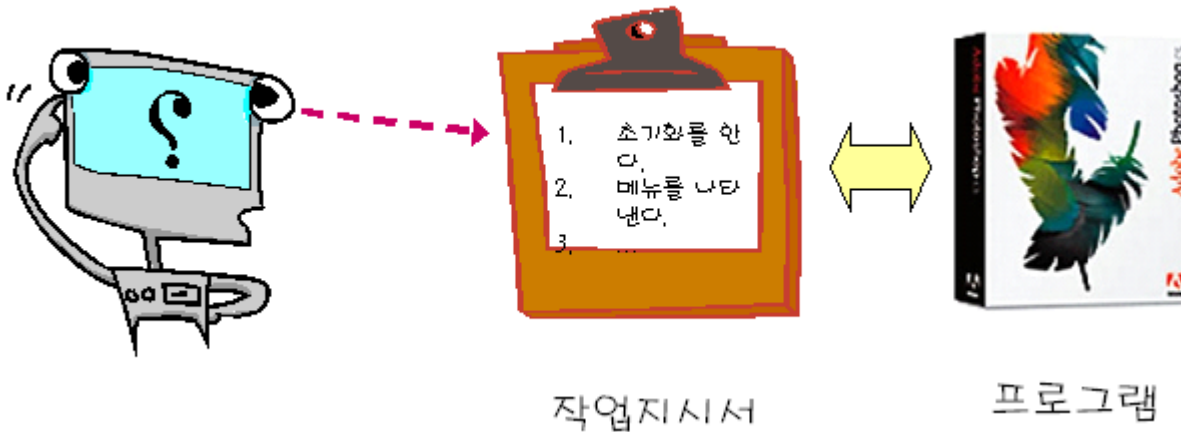


# 프로그램의 역할

Q) 컴퓨터에서 프로그램이 하는 일은 무엇인가?

A) 프로그램이란 우리가 하고자 하는 작업을 컴퓨터에게 전달하여 주는 역할을 한다.

I

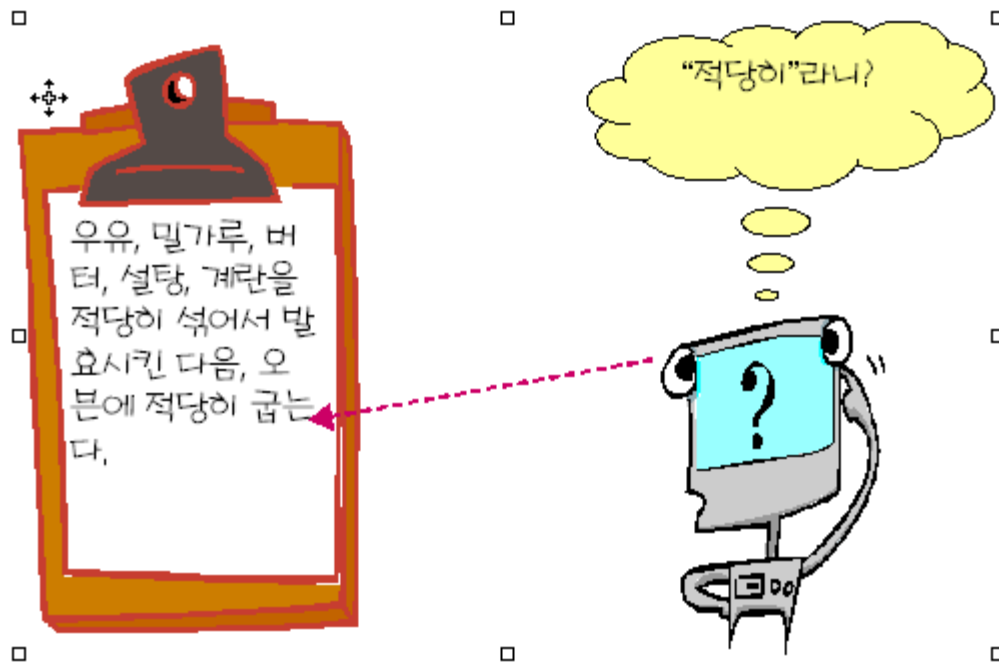




# 작업을 지시하는 방법

Q) 컴퓨터에게 어떻게 작업을 시킬 수 있을까?

A) 상식이나 지능이 없기 때문에 아주 자세하고 구체적으로 일을 지시하여야 한다.





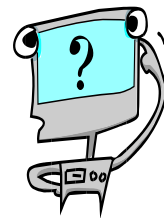
학생들의 성적  
의 평균을 계산  
해줘



"평균"이란  
의미를 모른다,



학생들의 성적을  
입력받고 입력받  
은 성적을 합하여,  
학생수로 나눈다,



이제 어느정도  
지시는 알겠지  
만 결과는 어떻  
게 하라는건지  
모르겠다,



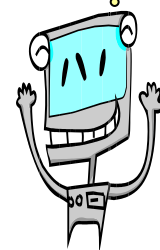
평균이란  
$$\frac{1}{n}(x_1 + x_2 + \dots + x_n)$$



단어의 의미는 알  
겠지만 단계적인  
절차를 말해주어  
야 한다,



학생들의 성적을  
입력받고 입력받  
은 성적을 합하여,  
학생수로 나눈다,  
결과는 화면에 표  
시하라,

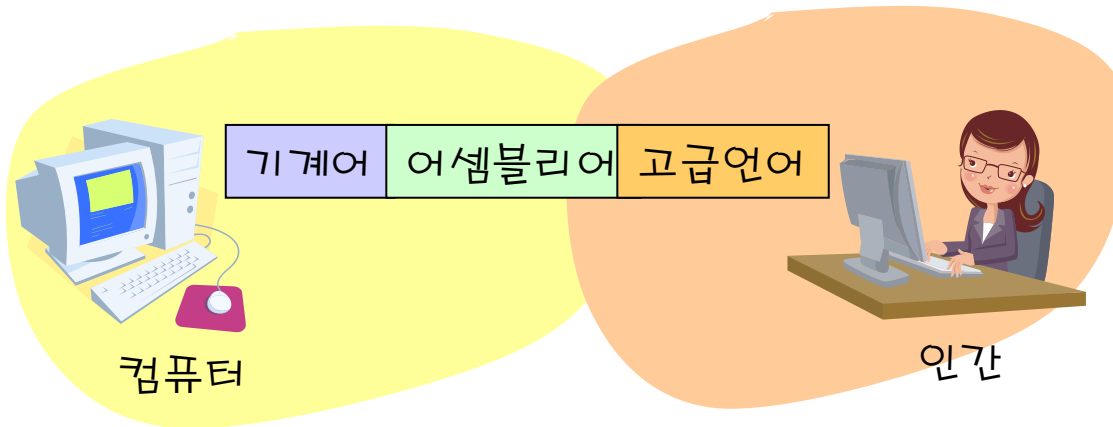


처음부터 이렇  
게 지시하였어  
야 한다,



# 프로그래밍 언어의 분류

- 기계어(machine language)
- 어셈블리어(assembly language)
- 고급 언어(high-level language)





# 기계어

- 특정 컴퓨터의 명령어(instruction)를 이진수로 표시한 것
- 0과 1로 구성
- 하드웨어에 종속

```
00001111 10111111 01000101 11111000
```

```
00001111 10111111 01001101 11111000
```

```
00000011 10100001
```

```
01100110 10001001 01000101 11111010
```





# 어셈블리어

- **CPU**의 명령어들을 이진수가 아닌 영어의 약자인 기호로 표기
- 기계어보다는 더 높은 수준에서 프로그램을 작성하는 것을 가능
- 기호와 **CPU**의 명령어가 일대일 대응
- 어셈블러(**assembler**): 기호를 이진수로 변환하는 프로그램

```
MOV AX, MIDSCORE  
MOV CX, FINALSORE  
ADD AX CX  
MOV TOTALSCORE, AX
```



# 고급언어

- 특정한 컴퓨터의 구조나 프로세서에 무관하게, 독립적으로 프로그램을 작성할 수 있는 언어
- **C, C++, JAVA, FORTRAN, PASCAL**
- 컴파일러: 고급 언어 문장을 기계어로 변환하는 프로그램

```
TotalScore = MidScore + FinalScore;
```



# 이번 장에서 학습할 내용



- 프로그램의 이해
- 프로그래밍 언어
- 알고리즘
- 프로그램 개발 과정

프로그램을  
작성하기에 앞서서  
중요한 개념들을  
살펴봅니다..





# 알고리즘

Q) 오븐의 사용법만 배우고 음식 재료만 있으면 누구나 요리가 가능한가?

A) 요리법을 알아야 한다.

- 프로그램이 요리와 같다면 알고리즘은 요리법에 해당한다.
- **알고리즘(algorithm)**: 문제를 해결하는 절차(방법)





# 빵을 만드는 알고리즘

- ① 빈 그릇을 준비한다.
- ② 이스트를 밀가루, 우유에 넣고 저어준다.
- ③ 버터, 설탕, 계란을 추가로 넣고 섞는다.
- ④ 따뜻한 곳에 놓아두어 발효시킨다
- ⑤ 170~180도의 오븐에서 굽는다



->



->



->



->





# 1부터 10까지의 합을 구하는 알고리즘

① 1부터 10까지의 숫자를 직접 하나씩 더한다.

$$I \quad 1 + 2 + 3 + \dots + 10 = 55$$

·② 두수의 합이 10이 되도록 숫자들을 그룹핑하여 그룹의 개수에 10을 곱하고 남은 숫자 5를 더한다.

$$(0 + 10) = 10$$

$$(1 + 9) = 10$$

$$(2 + 8) = 10$$

$$(3 + 7) = 10$$

$$(4 + 6) = 10$$

5

$$10 * 5 = 50$$

+

5

=

55

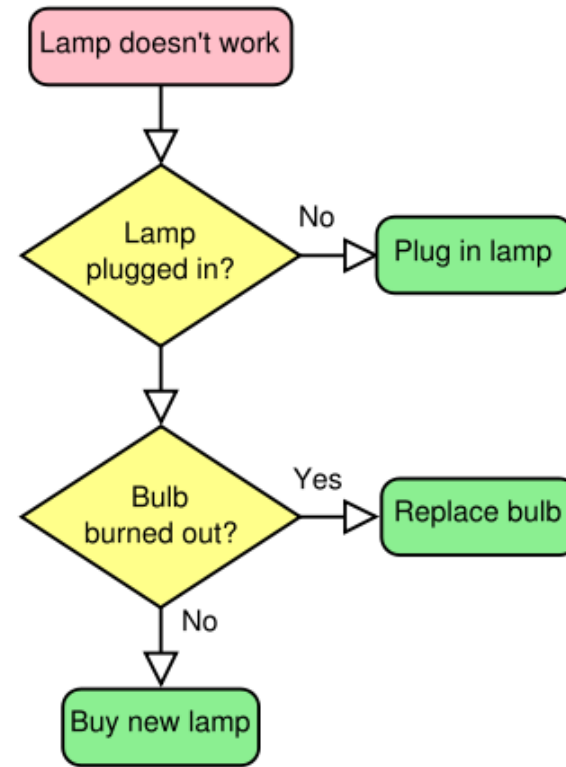
③ 공식을 이용하여 계산할 수도 있다.

$$10 * (1 + 10) / 2 = 55$$



# 알고리즘의 기술

- 자연어(natural language)
- 순서도(flowchart)
- 의사 코드(pseudo-code)





# 자연어

- 자연어 (**natural language**)는 인간이 사용하는 언어
- 단어들을 명백하게 정의해야 한다.

- 1.리스트의 첫 번째 숫자가 가장 크다고 가정하자.
- 2.리스트의 남아있는 숫자들이 하나씩 조사하여 현재의 최대값보다 크면 노트에 적는다.
- 3.모든 숫자들을 전부 조사된 후에 노트에 가장 나중에 적힌 숫자가 최대값이 된다.





# 알고리즘의 기술

- **순서도(flow chart):** 프로그램에서의 논리 순서 또는 작업 순서를 그림으로 표현하는 방법



수행의 시작/종료



처리



판단



연결자(흐름도가 너무 커서 한페이지에 나타낼수 없을때 어디로 가라는 표시에 사용)



입출력



# 알고리즘의 예

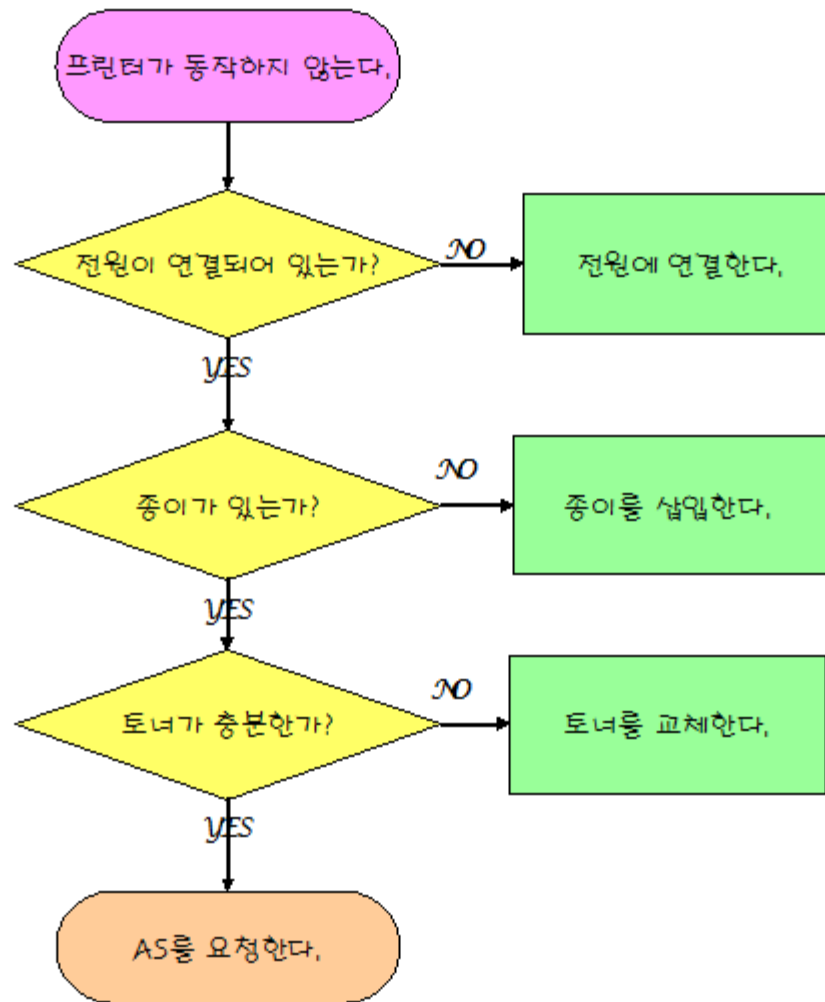


그림 1.14 순서도의 예: 프린터 고장을 처리하는 알고리즘



# 의사 코드

- **의사 코드(Pseudocode):** 자연어보다는 더 체계적이고 프로그래밍 언어보다는 덜 엄격한 언어로서 알고리즘의 표현에 주로 사용되는 코드

알고리즘 GetLargest

입력: 숫자들의 리스트 L.

출력: 리스트에서 가장 큰 값

```
largest ← L[0]
for each n in L do
  if n > largest then
    largest ← n
return largest
```



# 알고리즘을 만드는 방법

문제를 한 번에 해결하려고 하지 말고 더 작은 크기의 문제들로 분해한다.  
문제가 충분히 작아질 때까지 계속해서 분해한다.

- 방을 청소한다.
- 거실을 청소한다.
- 부엌을 청소한다.

- 환기를 시킨다.
- 물건들을 정리한다.
- 진공 청소기를 돌린다.
- 걸레질을 한다.



환기



물건 정리



진공청소기



걸레질



# 이번 장에서 학습할 내용



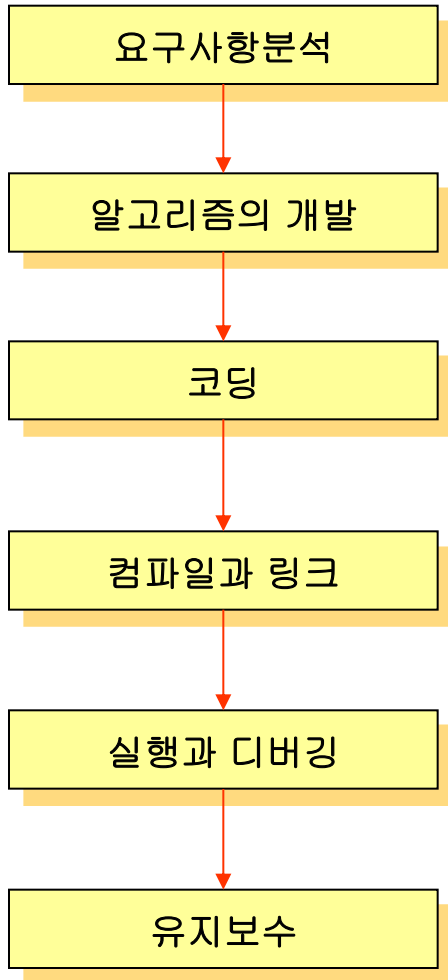
- 프로그래밍의 개념
- 프로그래밍 언어
- 알고리즘
- 프로그램 개발 과정

프로그램을  
작성하기에 앞서서  
중요한 개념들을  
살펴봅니다..





# 프로그램 개발 과정





# 요구 사항 분석

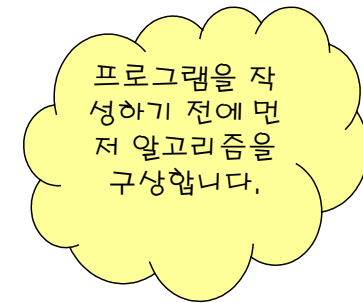
- 프로그래머는 사용자들의 요구사항을 만족시키기 위하여 프로그램을 작성
- (예) 3년 이상 근무한 직원들의 리스트 출력
  - 정규직만 **or** 계약직 포함
  - 기준이 되는 날짜가 오늘?
- 요구 사항 명세서: 사용자의 요구 조건을 만족하도록 소프트웨어가 갖는 기능 및 제약 조건, 성능 목표 등을 포함





# 알고리즘의 개발

- 핵심적인 부분
- 어떤 단계를 밟아서 어떤 순서로 작업을 처리할 것인지를 설계
- 순서도와 의사 코드를 도구로 사용
- 알고리즘은 프로그래밍 언어와는 무관
- 알고리즘은 원하는 결과를 얻기 위하여 밟아야 하는 단계에 집중적으로 초점을 맞추는 것







# 소스 작성

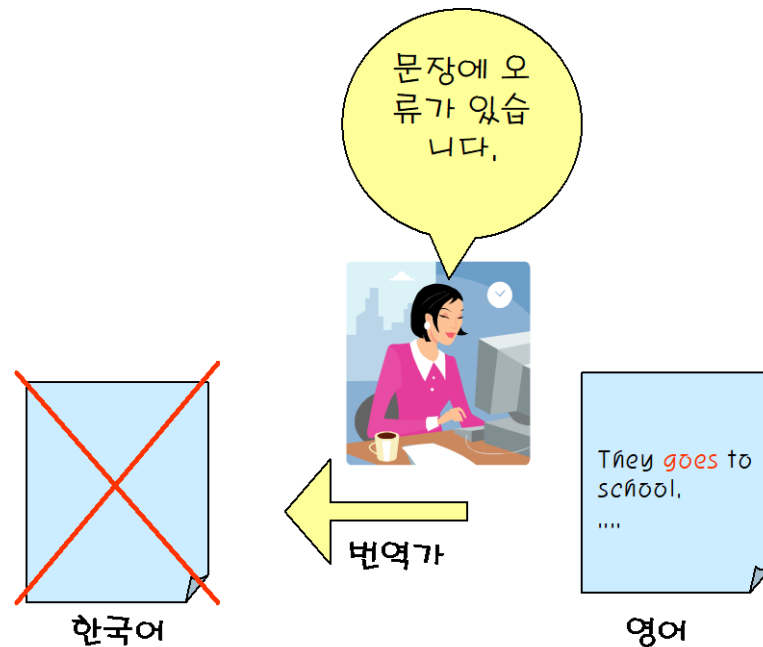
- 알고리즘의 각 단계를 프로그래밍 언어를 이용하여 기술
- 알고리즘을 프로그래밍 언어의 문법에 맞추어 기술한 것을 **소스 프로그램(source program)**
- 소스 프로그램은 주로 텍스트 에디터나 통합 개발 환경을 이용하여 작성
- 소스 파일 이름: (예) `test.c`





# 컴파일

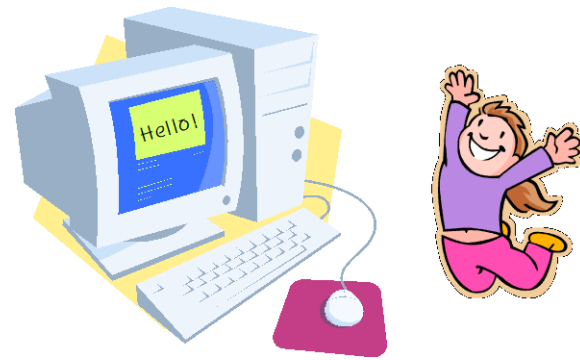
- 소스 프로그램을 오브젝트 프로그램으로 변환하는 작업
- 오브젝트 파일 이름: (예) `test.obj`
- 컴파일러가 수행
- 컴파일 오류(**compliance error**): 문법 오류
  - (예) `He go to school;`
- 오류가 발생하면 소스 프로그램을 수정한 후에 다시 컴파일





# 링크

- 컴파일된 목적 프로그램을 라이브러리와 연결하여 실행 프로그램을 작성하는 것
- 실행 파일 이름: (예) **test.exe**
- **라이브러리(library)**: 프로그래머들이 많이 사용되는 기능을 미리 작성해 놓은 것
  - (예) 입출력 기능, 파일 처리, 수학 함수 계산
- 링크를 수행하는 프로그램을 **링커(linker)**라고 한다.



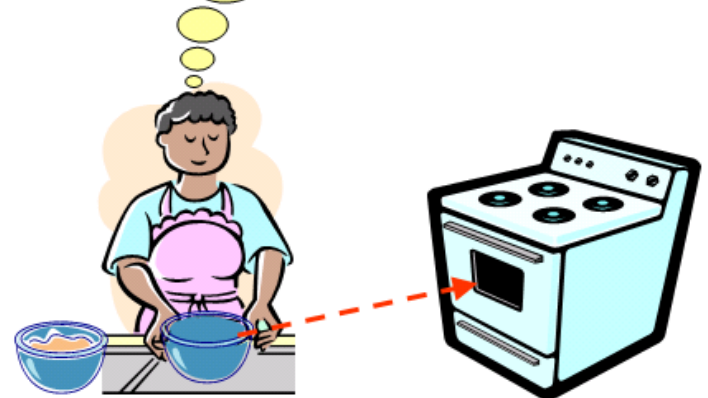


# 실행 및 디버깅

- 실행 시간 오류(run time error):
  - (예) 0으로 나누는 것
  - 잘못된 메모리 주소에 접근하는 것
- 논리 오류(logical error):
  - 문법은 틀리지 않았으나 논리적으로 정확하지 않는 것
  - (예)

- ① 그릇1과 그릇2를 준비한다.
- ② 그릇1에 밀가루, 우유, 계란을 넣고 잘 섞는다.
- ③ 그릇2를 오븐에 넣고 30분 동안 350도로 굽는다.

실수로 빈그릇  
을 오븐에 넣  
는다면 논리적  
인 오류입니다.





# 디버깅

- 소스에 존재하는 오류를 잡는 것



## 디버깅의 유래

1945년 마크 II라는 컴퓨터가 날아든 나방 때문에 고장을 일으켰고 이것을 "컴퓨터 버그(bug: 벌레)"라고 불렀다. 호퍼라는 사람이 나방을 채집해 기록에 남기고 이를 "디버깅(debugging)" 작업이라고 보고하였다. 이때부터 오류를 수정하는 작업을 디버깅이라고 부르기 시작하였다.

9/9

0800 Antism started  
1000 stopped - antism ✓ { 1.2700 9.032 117 045  
1300 (032) HP-MC 4.130476415-045 9.117 846 045 count  
032 PRO-2 2.130476415  
count 2.130476415  
Relays 6-2 in 032 failed special speed test  
in relay 10.00 sec -

1100 Relays changed  
1525 Started Cosine Tape (Sine check)  
Started Multi-Adder Test.

1545 Relay #70 Panel F  
(math) in relay.

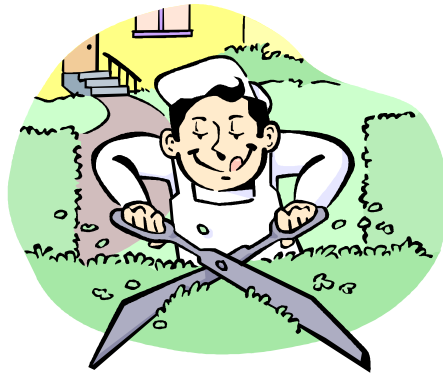
First actual case of bug being found.

1650 Antismant started.  
1700 closed down.



# 소프트웨어의 유지 보수

- 소프트웨어의 유지 보수가 필요한 이유
  - 디버깅 후에도 버그가 남아 있을 수 있기 때문
  - 소프트웨어가 개발된 다음에 사용자의 요구가 추가될 수 있기 때문
- 유지 보수 비용이 전체 비용의 **50%** 이상을 차지





# Q & A

