



## 2. 수학 함수를 C 함수로

- C는 함수(function)들의 집합입니다.
- C의 함수는 수학(mathematics)에서 사용하는 함수의 개념과 유사합니다.
- C로 프로그램을 코딩(coding)한다는 것은, 프로그램에서 필요로 하는 함수를 만들어 주는 것을 의미합니다.

- 
- 다음과 같은 4개의 수학함수가 주어졌을 때 이를 C의 함수로 바꾸는 과정을 살펴봅시다.

(1)  $f(x)=x^2+5$

(2)  $g(x,y)=f(x)+y$

(3)  $y=3$

(4)  $l(x,y)=\{\text{원점에서 } (x,y)\text{까지 선을 그린다}\}$

---

## 용어 정의

- 함수  $f(x)=x^2+5$  에서  $f$ 는 **함수 이름(function name)**입니다.
- 함수가 받는 **파라미터(parameter)**는  $x$ 인데, 함수 이름 다음에 괄호를 써서 나타냅니다.
- 함수가 하는 일은 등호(=) 다음에 정 의하는데, 이를 **함수 몸체(function body)**라고 합니다.
- 함수를 **정의(define)**하는 쪽의 파라미터를 **형식 파라미터(formal parameter)**라고 합니다.

- 
- 함수  $f(x)=x^2+5$

$$x=2$$

$$y=3$$

$$i=f(y)$$

- 위와 같은 일련의 문장이 실행되었을 때,  $i$ 의 값은 얼마일까요?

$$f(x)=x^2+5$$

**1단계.** 함수의 이름과 파라미터를 그대로 써 줍니다.

$$\underline{f(x)}=x^2+5 \rightarrow f(\quad x)$$

**2단계.** 함수의 몸체 부분을 여는 브레이스(open brace: {})와 닫는 브레이스(close brace: {})안에 써 줍니다■.

$$\underline{f(x)}=x^2+5 \rightarrow f(\quad x) \{ \quad x*x+5 \quad \}$$

f(    x)

{

  x\*x+5  

}

**3단계.** 함수가 리턴(return)하는 값이 무엇인지를 명시(specify)합니다.

```
f(    x)
{
    t=x*x+5
    return t
}
```

- "C 언어"를 통해 표현할 때 컴퓨터가 문장의 끝을 알 수 있도록 특정한 표현을 해 주어야 합니다■.

```
f(    x)
{
    t=x*x+5;
    return t;
}
```

---

**4단계.** 파라미터 변수와 함수 안에서 사용된 변수의 형을 선언합니다.

```
f(int x)
{
    int t;
    t=x*x+5;
    return t;
}
```

- **int**는 '변수가 메모리를 4바이트 차지하면서, 표현은 정수(integer)만이 허용된다'라는 의미입니다.

---

**5단계.** 함수의 리턴형을 선언합니다.

```
int f(int x)
{
    int t;
    t=x*x+5;
    return t■;
}
```





## 아규먼트(argument)와 파라미터(parameter)

- 아규먼트는 함수를 호출하는 쪽에서의 **실 인자(actual parameter)**를 가리키는 말이며, 파라미터는 함수를 정의하는 쪽에서의 **형식 인자(formal parameter)**를 가리키는 말입니다.

```
...  
void F(int k) {  
    //어떤 일을 함  
    //재미있는 일도 함.  
    //k와 관계된 어떤 일도 함  
}  
...  
void main() {  
    int i=10;  
    F(i);  
}  
..
```

- 위의 예에서 main()의 F(i)에서 i의 값인 10을 - 의미가 명확하다면 i를 - 실 인자라고 합니다
- 함수를 정의하는 쪽에서의 k를 형식 인자라고 합니다(실 인자를 받기 위해 형식적으로 적어둔 변수입니다).



## 관례(convention)

- int, return 등은 언어에 의해서 미리 정의된 **예약어(reserved word)**입니다. 이러한 예약어를 **키워드(keyword)**라고도 합니다.
- 변수 이름 x, t나, 함수 이름 f 등은 우리가 - 규칙에 맞다면 - 마음대로 정할 수 있습니다. 이러한 사용자에게 의해서 정의되는 단어를 **명칭(identifier)**이라고 합니다.
- 위의 함수 f는 아래와 같이 작성해도 같은 역할을 합니다.

```
int MyFirstFunction(int parameterX)
{
    int temporary;

    temporary=parameterX*parameterX+5;
    return temporary;
}
```

- 사용자 함수 이름은 대문자로 변수 이름은 소문자로 시작하도록 정하도록 합시다.
- 변수 이름을 정할 때, 변수의 역할을 이해하기 쉽도록 접두어(prefix)를 붙이는데, 예를 들면, 아래와 같습니다.

nFileOpened  
xCurrent

- **변수의 선언 문장과 실행 문장 사이를 한 줄 띄웁니다**

```
int f(int x)
{
    int t;

    _____
    t=x*x+5;
    return t;
}
```

- **들여쓰기(indentation)**를 하면 후에 설명할 제어문(control statement)등에서 문장의 포함관계를 쉽게 알 수 있습니다.

```
int f(int x)
{
    ____int t;

    ____t=x*x+5;
    ____return t;
}
```

- 
- 한 문장은 되도록이면 한 줄에 적습니다.

```
int f(int x){int t;t=x*x+5;return t;}
```

- 위와 같이 적어도 무방합니다.

```
int f(int x) {  
    int t;  
  
    t=x*x+5;  
    return t;  
}
```

---

$$g(x,y)=f(x)+y$$

```
int g(int x, int y) {  
    int t;  
  
    t=f(x)+y;  
    return t;  
}
```

- 함수의 몸체를 정의(define)할 때는 이미 정의된 다른 함수를 사용 - 호출 - 할 수 있습니다.
- 위에서 f()가 정의되었으므로 위의 함수 g() 정의는 타당합니다.
- 아직 정의되지 않은 함수 h()는 호출할 수 없습니다.

---

```
int g(int x, int y) {  
    int t;  
  
    t=f(x)+y;  
    return t;  
}
```

- 파라미터가 2개 이상일 때는 콤마(comma: ,)로 구분을 합니다. 이것을 **파라미터 리스트 (parameter list)**라고 합니다.



---

$y=3$

```
int y(void) {  
    return 3;  
}
```

- 함수가 파라미터를 받지 않는 경우, **파라미터가 없다**라는 표현을 **void**로 합니다.

$l(x,y)=\{\text{원점에서 } (x,y)\text{까지 선을 그린다}\}$

```
void l(int x,int y) {  
    line(0,0,x,y);  
}
```



line()은 그래픽의 그리기 함수라고 가정합니다. 이렇게 미리 만들어진 함수(built in function)중, 모든 컴파일러에서 지원하도록 규정된 함수들의 모임을 표준 함수(standard function)라고 합니다. 초창기에 ANSI에서 정한 표준 C함수의 수는 100여 개 뿐이었습니다. C++에는 수많은 표준함수와 표준 클래스 및 표준 객체들이 존재합니다.

- 이렇게 **함수의 리턴값이 없다**라는 표현도 void로 합니다.

## main()

- “시작하는 함수는 main()입니다. 반드시 그리고 유일하게 1개 있어야 합니다.”
- 실행프로그램이 운영체제에 의해서 로드(load)된 후 운영체제는 제일 먼저 main()을 호출합니다(플랫폼platform이 Win32의 경우, 운영체제는 WinMain()을 호출합니다. 하지만, 대부분의 운영체제에서는 main()이 시작하는 함수입니다).
- 그러므로 실행 파일을 만드는 소스마다 반드시 1개만의 main() 함수를 가져야 합니다.
- main()의 원형은 여러 개가 존재합니다. 일반적으로는 다음과 같습니다.

```
void main(void)
```



## 함수도 선언해야 한다.

```
void main() {  
    int i;  
  
    i=f(3);  
}  
int f(int x) {  
    int t;  
  
    t=x*x+5;  
    return t;  
}
```

- 소스는 다음과 같이 수정되어야 합니다.

```
int f(int x);  
void main() {  
    int i;  
  
    i=f(3);  
}  
int f(int x) {  
    int t;  
  
    t=x*x+5;  
    return t;  
}
```

- 다음과 같이 소스를 수정할 수 있습니다.

```
int f(int x) {  
    int t;  
  
    t=x*x+5;  
    return t;  
}  
void main() {  
    int i;  
  
    i=f(3);  
}
```

- 위의 스타일에서는 함수 f()에 관한 선언(declaration)은 없어도 됩니다

- 
- 컴파일러는 ANSI에서 정한 표준 함수(standard function)를 미리 만들어서 제공하는데, 이를 **표준 함수**라고 합니다.

printf()

- 이 함수는 **표준 출력(standard output)**이라고 불리는 **파일(file)**에 **문자열(character string)**을 출력합니다.



문자는 ASCII의 1문자를 의미하며, A인 경우 C언어에서 'A'처럼 표현합니다. 문자열은 0개 이상의 문자의 연속한 순서를 의미합니다. ABC라는 문자열은 "ABC"로 표시합니다. 특별히 길이가 0인 ""는 빈 문자열(empty string)이라고 합니다.

- 화면(screen)의 현재 커서(cursor) 위치에 문자열 "I love the God"를 출력합니다.

printf("I love the God");

- 이 프로그램을 C로 만들어 봅시다.

```
void main() {  
    printf("I love the God");  
}
```



- 위의 프로그램은 아직 부족합니다. '함수는 쓰기 전에 선언해야 합니다.'라는 중요한 규칙을 만족하지 못하고 있기 때문입니다. printf()의 선언문이 필요한 것입니다.



---

## 끼워 넣기include

stdio.h : 표준 입출력에 관한 함수의 선언이 들어 있습니다.

stdlib.h : 표준 함수 중 입출력 외의 함수들이 들어 있습니다.

math.h : 표준 수학 함수의 선언이 들어 있습니다.

- 디스크에 별도로 존재하는 이러한 외부 파일을 끼워 넣는 명령문은 다음과 같습니다.

#include

- printf()는 표준 출력에 사용되는 함수이므로, stdio.h에 선언이 들어 있습니다.

---

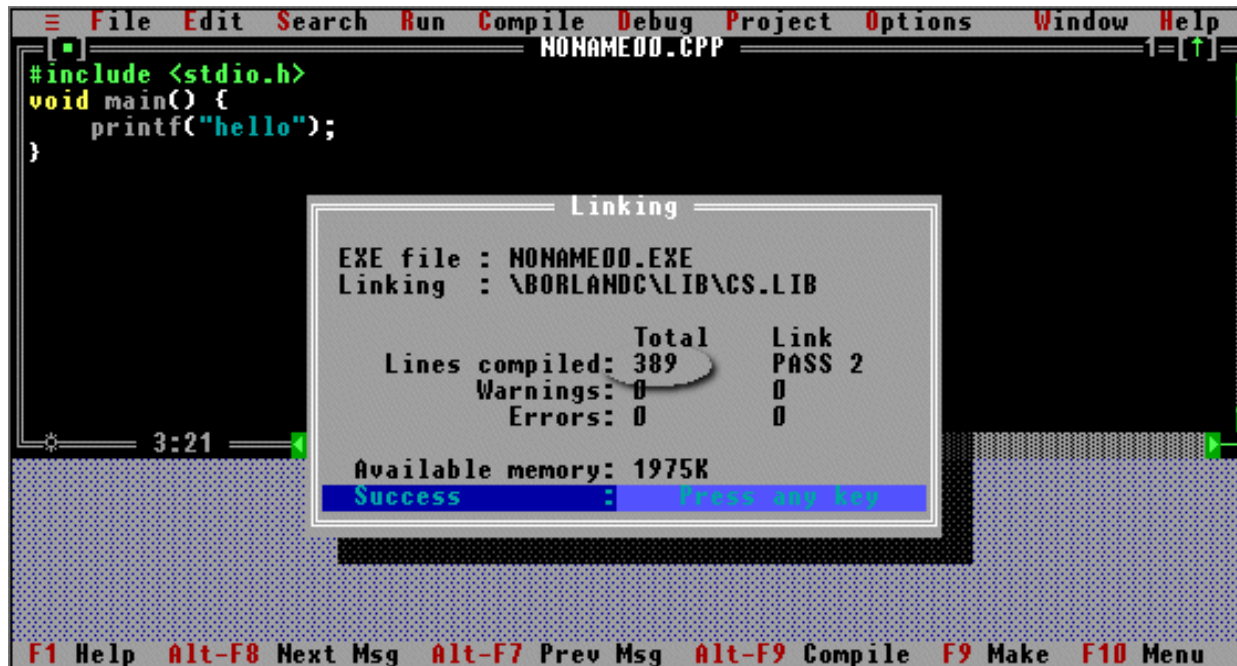
```
#include <stdio.h>
```

```
void main() {  
    printf("I love the God");  
}
```

- 헤더 파일의 이름을 작다(less than: <)와 크다(greater than: >) 기호 사이에 적어줍니다.

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

- 전처리(preprocessing) 명령문, 혹은 컴파일러 지시자(compiler directive)



```
#include <stdio.h>
void main() {
    printf("hello");
}
```

Linking

EXE file : NONAME00.EXE  
Linking : \BORLANDC\LIB\CS.LIB

	Total	Link
Lines compiled:	389	PASS 2
Warnings:	0	0
Errors:	0	0

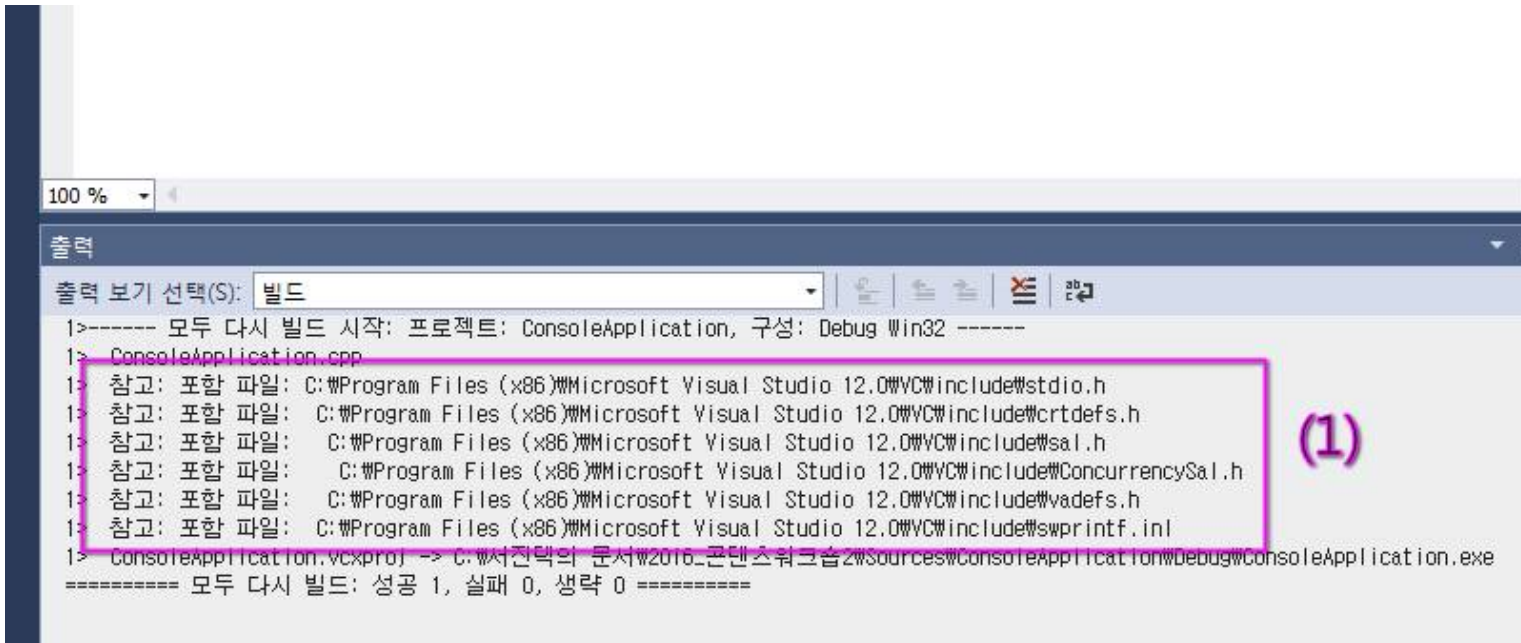
Available memory: 1975K  
Success : Press any key



Borland C++에서 컴파일 화면: 원래의 소스는 4줄이지만, `#include <stdio.h>`에 의해 컴파일된 소스가 389줄인 것을 확인할 수 있습니다.

- 
- C에서 전처리 명령문은 모두 특수 문자 `#` 으로 시작합니다.
  - 많이 사용하는 전처리 명령문에는 `#define`, `#ifdef`, `#endif` 등이 있습니다.





```

100 %
출력
출력 보기 선택(S): 빌드
1>----- 모두 다시 빌드 시작: 프로젝트: ConsoleApplication, 구성: Debug Win32 -----
1> ConsoleApplication.cpp
1> 참고: 포함 파일: C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\include\stdio.h
1> 참고: 포함 파일: C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\include\crtdefs.h
1> 참고: 포함 파일: C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\include\saf.h
1> 참고: 포함 파일: C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\include\ConcurrencySal.h
1> 참고: 포함 파일: C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\include\vadefs.h
1> 참고: 포함 파일: C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\include\swprintf.inl
1> ConsoleApplication.vcxproj -> C:\서전택의 문서\2016_콘텐츠워크숍2\Sources\ConsoleApplication\Debug\ConsoleApplication.exe
===== 모두 다시 빌드: 성공 1, 실패 0, 생략 0 =====
  
```



프로젝트를 빌드하면, ConsoleApplication.cpp를 빌드할 때 포함된 파일들의 목록을 확인할 수 있습니다.

• 이제 시작입니다!



## 소스 문자 집합(source character set)

**a b c d e f g h i j k l m n o p q r s t u v w x y z**  
**A B C D E F G H I J K L M N O P Q R S T U V W X Y Z**  
**0 1 2 3 4 5 6 7 8 9**  
**\_ { } [ ] # ( ) < > % : ; . ? \* + - / ^ & | ~ ! = , \ " ' "**

- 위의 문자들은 변수(variable), 명칭(identifier), 숫자(number), 기호(symbol)등을 형성합니다.
- 이러한 것들을 모두 토큰(token)이라고 합니다. 토큰은 컴파일러가 코드를 생성하는데 필요한 의미 있는 단위를 말합니다.

- 영문자와 숫자를 제외한 **특수문자(구분자: delimiter)**의 발음은 아래와 같습니다.

_	underscore(밑줄)
{	open brace or open curly bracket(여는 대괄호)
}	close brace(닫는 대괄호)
[	open bracket(여는 대대괄호)
]	close bracket(닫는 대대괄호)
#	pound(sharp or number)
(	open parenthesis(여는 괄호)
)	close parenthesis(닫는 괄호)
<	less than(보다 작다)
>	greater than(보다 크다)
%	percent
:	colon
;	semicolon
.	period(점)



---

?	question mark(물음표)
*	asterisk or star(별표)
+	plus
-	minus
/	divide
^	caret
&	ampersand
	vertical bar(수직 바)
~	tilde(틸더, 물결표)
!	exclamation mark(느낌표)
=	equal
,	comma
\	back slash(역 슬래쉬)
"	double quotation mark(이중 인용 부호)
'	single quotation mark(인용 부호)