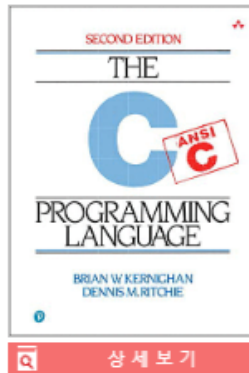


Lecture 01

C 프로그래밍 소개

교재



자료유형 : 서양서단행본
서명 / 저자 : The C Programming Language / Brian W. Kernighan Dennis M. Ritchie
개인저자 : [Brian W. Kernighan](#)
[Ritchie Dennis M](#)
판사항 : Second Edition
발행사항 : United States : PH PTR, 1988.
형태사항 : xii, 272 p. ; 24cm.
일반사항 : Includes index
주제명 : [C Programming Language](#)
[C Programming](#)
[C Language](#)
ISBN : 0131103628
청구기호 : 005.13 B849c



자료유형 : 동양서단행본
서명 / 저자 : (Kernighan의) C 언어 프로그래밍 / Brian W. Kernighan, Dennis M. Ritchie 지음 ; 김석환, 박용규, 최홍순 옮김
원서명 : [The C programming language](#) [C programming language](#)
개인저자 : [Kernighan, Brian W](#)
[Ritchie, Dennis M., 1941-2011](#)
[김석환](#)
[박용규](#)
[최홍순](#)
발행사항 : 서울 : 휴먼사이언스, 2016
형태사항 : xiv, 389 p. ; 삽화 ; 26 cm
일반사항 : 부록: A. 참조 매뉴얼 -- B. 표준 라이브러리 -- C. 개선점 요약
색인수록
웹자원정보 : Table of Contents: http://www.riss.kr/Keris_abstoc.do?no=14047190
ISBN : 9788993712674
청구기호 : 005.133 K39c 7

C 언어

- Dennis Ritchies – AT&T Bell 연구소 – 1972

- 16-bit DEC PDP-11 컴퓨터
- 영향을 받은 언어
 - B, 포트란, 어셈블리어, ...
- 영향을 준 언어
 - C++, C#, 자바, 줄리아, ...

- 널리 사용됨

- 호환성
- 성능(처리 속도, 메모리, ...)
- Low-level access



By Kozan - Own work, Public Domain,
<https://commons.wikimedia.org/w/index.php?curid=47961466>

C 언어의 특징

- C 언어의 특징
 - 키워드 많지 않음
 - 구조화 된 데이터 타입(struct, union)
 - 포인터
 - 라이브러리
 - `stdio.h` – C 언어의 입출력 제공
 - `math.h` – 수학 함수 제공
 - `string.h` – 문자열 조작 함수 제공
 - ...
 - 머신 코드로 컴파일 할 수 있음
 - Macro preprocessor

C 언어의 버전

- 1972 – C 개발됨
- 1978 – “The C Programming Language” 책 출판됨
- 1989 – C89 표준(ANSI C 혹은 Standard C로 널리 알려짐)
- 1990 – ANSI C가 ISO/IEC 9899:1990 명칭으로 출간됨
C90로 널리 알려짐
- 1999 – C99 표준
- 2011 – C11 표준
- 2018 – C17 표준

- 본 수업에서 **ANSI C(C89/C90)** 사용
-

C 언어의 실용

■ 시스템 프로그래밍

- Linux와 같은 운영체제 프로그래밍
- 마이크로 컨트롤러: 가전제품, IoT 기기, 자동차, 비행기, ...
- 임베디드 프로세서: 스마트폰, 웨어러블 기기, ...
- DSP 프로세서: 스피커, TV, ...
- ...



C 언어 vs. 다른 언어

- C 언어 기반으로 개발됨: C++, Objective C, C#
- C 언어에 영향을 받음: 자바, 펄, 줄리아, 파이썬, ...
- C 언어에 부족한 것
 - Exception
 - Range-checking
 - Garbage collection
 - Object-oriented programming
 - Polymorphism
 - ...
- Low-level 프로그래밍 언어 → 크기가 작고 빠른 코드(**보통**)

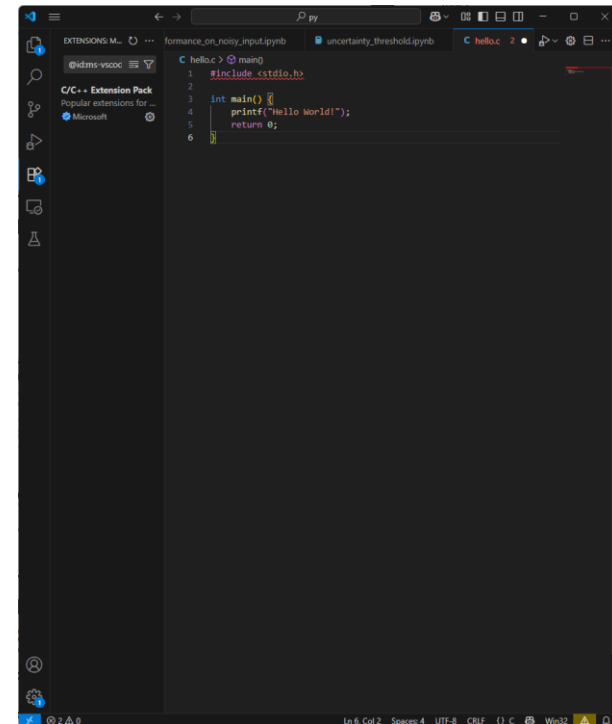
C 코드 작성

■ .c 및 .h 파일 확장자



```
1 #include <stdio.h>
2
3 int main() {
4     printf("Hello World!");
5     return 0;
6 }
```

The screenshot shows a Notepad++ window with a C program. The code is as follows:



```
1 #include <stdio.h>
2
3 int main() {
4     printf("Hello World!");
5     return 0;
6 }
```


The screenshot shows a Visual Studio Code window with a C program. The code is as follows:

C 코드 컴파일

- gcc 컴파일러 사용

gcc -Wall **infilename.c** -o **outfilename**

예, gcc -Wall **helloworld.c** -o **helloworld**





```
Microsoft Windows [Version 10.0.26100.4349]
(c) Microsoft Corporation. All rights reserved.

C:\Users\USER>cd /d C:\Users\USER\Dropbox\KNUT_lectures\c_programming\examples\hello_world

C:\Users\USER\Dropbox\KNUT_lectures\c_programming\examples\hello_world>gcc -Wall helloworld.c -o helloworld

C:\Users\USER\Dropbox\KNUT_lectures\c_programming\examples\hello_world>helloworld
Hello World!
C:\Users\USER\Dropbox\KNUT_lectures\c_programming\examples\hello_world>|
```

| 이름 | 상태 | 수정한 날짜 | 유형 | 크기 |
|--|----|--------------------|---------|-------|
|  helloworld.c | ✓ | 2025-06-29 오후 4:51 | C 파일 | 1KB |
|  helloworld.exe | ✓ | 2025-06-29 오후 4:55 | 응용 프로그램 | 129KB |

C 코드 디버깅

■ gdb 사용

gcc -Wall **infilename.c** -o **outfilename**

gdb **outfilename**

```
명령 프롬프트
C:\Users\USER\Dropbox\KNUT_lectures\c_programming\examples\hello_world>gcc -Wall helloworld.c -o helloworld
C:\Users\USER\Dropbox\KNUT_lectures\c_programming\examples\hello_world>gdb helloworld
GNU gdb (GDB) 15.2
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-w64-mingw32".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from helloworld...
(gdb) run
Starting program: C:\Users\USER\Dropbox\KNUT_lectures\c_programming\examples\hello_world\helloworld.exe
[New Thread 7668.0x44e0]
Hello World! [Thread 7668.0x44e0 exited with code 0]
[Inferior 1 (process 7668) exited normally]
(gdb) q

C:\Users\USER\Dropbox\KNUT_lectures\c_programming\examples\hello_world>
```

IDE(Integrated Development Environment)



- 코드 작성, 컴파일, 디버그 등은 통합 환경에 실행 가능
- 큰 프로그램 개발 시 적합함
- 널리 사용되는 IDE
 - Eclipse(<https://www.eclipse.org/downloads/>)
 - Microsoft Visual Studio(<https://visualstudio.microsoft.com/ko/>)
 - Xcode(<https://developer.apple.com/xcode/>)
 - Kdevelop(<https://kdevelop.org/>)
 - ...
- IDE 사용 시 유의 사항
 - 보통 IDE 첫 사용 시 컴파일러와 디버거를 따로 설정해 야 함
 - Debug와 Release 모드 사이의 차이

첫 프로그램

- Hello World!
- .c 파일의 구조
- 문법: 코멘트, macro, 변수/함수 선언
- main() 함수 및 함수의 구조
- 표현, 연산의 처리 순서
- 기본 I/O

.c 파일의 구조

```
/* 코드 설명을 위한 코멘트
   이는 프로그램의 실행에 아무 영향을 미치지 않음
*/
```

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

```
// 함수의 프로토타입 선언
```

```
int square(int a);
```

```
// 변수 선언
```

```
int x = 10;
```

```
// main() 함수 선언
```

```
int main() {
    printf("%d\n", square(x));
    return 0;
}
```

```
// 함수의 선언
```

```
int square(int a) {
    return a*a;
}
```

코멘트(설명문)

- 코멘트: /* 난 코멘트다 */
 - 여러 행을 차지할 수 있음
 - 코멘트는 /*와 */ 사이에 있는 모든 문자임
- 단일 행 코멘트: // 난 단일 행 코멘트다
- 코멘트는 프로그램의 실행에 아무 영향을 미치지 않음
- 프로그램의 중간중간에 코멘트를 넣는 것 **좋은 습관**임
 - 프로그램을 이해하기에 큰 도움이 됨
 - 향후 프로그램을 쉽게 관리할 수 있음

#include macro

- .h 파일 확장자
 - 헤더 파일(header file)
 - 헤더 파일에 상수, 함수 등의 선언은 포함됨
- #include <stdio.h>
 - `stdio.h`는 표준 라이브러리라고 함
 - `stdio.h`에 선언된 기본 I/O 함수들을 모두 읽어 넣음
 - 코드를 컴파일할 때 표준 라이브러리를 추가하기는 기본값으로 설정됨
예, `printf()` 함수는 `stdio.h`에 선언되며, 코드를 컴파일할 때
“`gcc -Wall infilename.c -o outfilename`”만 해도 컴파일러가 알아서 `stdio.h`
표준 라이브러리에 `printf()` 함수의 선언을 찾아내고 컴파일을 함
- 사용자가 헤더 파일을 작성하여 사용할 수 있음 헤더 파일 경로
`gcc -Wall infilename.c -o outfilename -I<header_path>`
I의 대문자

변수의 선언

- 모든 변수는 사용하기 전에 선언해 야 함
 - `int a;`
 - `int m, n;`
 - `float pi;`
- `int`는 변수의 형이 정수형(integer)이라는 것을 알려줌
- `float`는 부동소수점형(floating-point)에 해당함
- `char` 문자형, 1바이트
- `short` 단정도 정수형
- `long` 배정도 정수형
- `double` 배정도 부동소수점형

변수의 초기화

- 전역 변수(global variable) 및 지역 변수(local variable)
 - 전역 변수: 모든 함수 밖에 선언됨
 - 지역 변수: 어느 함수 속에 선언됨
- 변수의 초기 값
 - 전역 변수: 0
 - 지역 변수: 쓰레기(garbage) 값
- 변수를 선언할 때 변수의 초기 값을 같이 지정해 줄 수 있음
 - `int a = 10;`
 - `int a, b = 3, c = 8;`
 - `float pi = 3.14159;`

```
#include <stdio.h>

int a; // 전역 변수

void func() {
    int a; // 지역 변수
    printf("func a = %d\n", a);
}

int main() {
    int b; // 지역 변수
    func();
    printf("a = %d\n", a);
    printf("b = %d\n", b);
    return 0;
}
```

수식 표현

- x 와 y 는 변수라고 가정함
 - Binary 수식: $x+y$, $x-y$, $x*y$, x/y , $x\%y$
 - 단순한 프로그램문: $y = x+3*x/(y-2);$
 - 수식에 의해 계산하여 지정하는 문: $x+=y$, $x-=y$, $x*=y$, $x/=y$, $x\%=y$

| 계산하여 지정하는 문 | 해당하는 수식 |
|-------------|-------------|
| $x+=y;$ | $x = x+y;$ |
| $x-=y;$ | $x = x-y;$ |
| $x*=y;$ | $x = x*y;$ |
| $x/=y;$ | $x = x/y;$ |
| $x\%=y;$ | $x = x\%y;$ |

연산의 처리 순서

■ 연산의 처리 순서

| 연산자 | 처리 순서 |
|---|---------------|
| $+, -$ (부호) | right-to-left |
| $*, /$, $\%$ | left-to-right |
| $+, -$ | left-to-right |
| $=$, $+=$, $-=$, $*=$, $/=$, $\%=$ | right-to-left |

- 연산의 처리 순서에 의존하는 것이 **좋지 않은 습관**임
- 괄호를 사용하여 처리 순서를 지정하는 것이 **좋은 습관**임

연산의 처리 순서

- $x=3.0$, $y=5.0$ 가정하여 다음 프로그램문을 실행하시오
`float z = x+5*x/(y-1);`

연산의 처리 순서

- $x=3.0$, $y=5.0$ 가정하여 다음 프로그램문을 실행하시오

float $z = x + 5 * x / (y - 1);$

1. 괄호를 먼저 실행함: **float** $z = x + 5 * x / 4.0;$

연산의 처리 순서

- $x=3.0$, $y=5.0$ 가정하여 다음 프로그램문을 실행하시오

float $z = x + 5 * x / (y - 1);$

1. 괄호를 먼저 실행함: **float** $z = x + 5 * x / 4.0;$
2. 곱하기와 나누기를 왼쪽에서 오른쪽으로 실행함

float $z = x + 15.0 / 4.0; \rightarrow$ **float** $z = x + 3.75;$

연산의 처리 순서

- $x=3.0$, $y=5.0$ 가정하여 다음 프로그램문을 실행하시오

float $z = x + 5 * x / (y - 1);$

1. 괄호를 먼저 실행함: **float** $z = x + 5 * x / 4.0;$
2. 곱하기와 나누기를 왼쪽에서 오른쪽으로 실행함
float $z = x + 15.0 / 4.0; \rightarrow$ **float** $z = x + 3.75;$
3. 더하기를 실행함: **float** $z = 6.75;$

연산의 처리 순서

- $x=3.0$, $y=5.0$ 가정하여 다음 프로그램문을 실행하시오

float $z = x + 5 * x / (y - 1);$

1. 괄호를 먼저 실행함: **float** $z = x + 5 * x / 4.0;$
2. 곱하기와 나누기를 왼쪽에서 오른쪽으로 실행함
float $z = x + 15.0 / 4.0; \rightarrow$ **float** $z = x + 3.75;$
3. 더하기를 실행함: **float** $z = 6.75;$
4. 변수를 초기화함: 이제 z 변수에 6.75 값을 지정함

$z = 4.5$ 를 얻기 위해 괄호를 어디에 추가하면 될까요?

float $z = (x + 5 * x) / (y - 1);$

함수의 프로토타입

- 함수도 호출하기 전에 선언해 야 함
- 함수의 선언이 함수의 프로토타입이라고 불림
- 예, `int square(int a);` 또는 `int square(int);`
- 자주 쓰는 함수들은 C의 표준 라이브러리에서 선언됨
- 함수의 일반적인 형태

```
리턴값의 형 [space] 함수 이름(매개변수) {  
    선언문;  
    문장들;  
}
```

함수의 프로토타입

■ 함수의 매개변수

- 함수를 호출할 때 쓰는 매개변수를 argument 또는 actual argument라고 불림
- 함수 내에서는 매개변수를 parameter 또는 formal argument라고 불림

■ 리턴값

- **return** 문을 통해 함수의 연산 결과를 나타내는 값
- **return**의 뒤에 변수가 올 수도 있고, 수식이 올 수도 있음
- 아무 값을 출력하지 않는 함수도 있음

■ **void**

- 아무 값을 출력하지 않는 함수를 나타냄
void func1 (**int**) ;
- 아무 매개변수를 받지 않는 함수를 나타냄
int func2 (**void**) ; **void** func3 (**void**) ;

main () 함수

- `main()` 함수는 C 프로그램의 주프로그램이라고 불림
 - 이름은 원하는 대로 붙일 수 없고, 꼭 `main`이라고 해 줘야 함
- 제일 간단한 버전: `int main(void);`
 - 입력 매개변수를 받지 않음
 - 0을 나타내는 경우 `main()` 함수가 정상 동작임
 - 0이 아닌 값을 나타내는 경우 에러가 발생한다는 의미
- 매개변수를 받는 버전
 - `int main(int argc, char** argv);`
 - Command-line argument라고 불림
 - `argc`: argument 개수
 - `argv`: 사용자 제공하는 argument 배열

함수의 선언

- 함수의 선언

```
리턴값의 형 [space] 함수 이름(매개변수) {  
    선언문;  
    문장들;  
}
```

- 함수의 선언이 함수의 프로토타입과 동일해 야 함
 - 매개변수 이름이 달라질 수 있음
 - 중괄호({})가 **코드 블록**이라고 불림
 - 블록에서 선언된 변수들은 그 블록에서만 사용할 수 있음

Hello World!

- `puts()` 함수
 - 콘솔 창(console window)에 문자열을 출력하고 커서(cursor)를 다음 행의 처음으로 보냄
- 스트링 상수(string literal)
 - "" 사이에 있는 문자들은 스트링 상수라고 불림
- `return 0;`
 - 함수가 정상적으로 끝남

```
#include <stdio.h>

int main() {
    puts("Hello World!");
    return 0;
}
```

Hello World!

- 스트링 상수를 변수에 저장한 뒤 출력

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

```
int main() {  
    const char msg[] = "Hello World!";  
    puts(msg);  
    return 0;  
}
```

- `const` 키워드: 선언한 변수를 상수라고 지정함
- `char`: 문자형, 예: 'a', 'b', '3'
- `const char msg[]`: 상수인 문자열

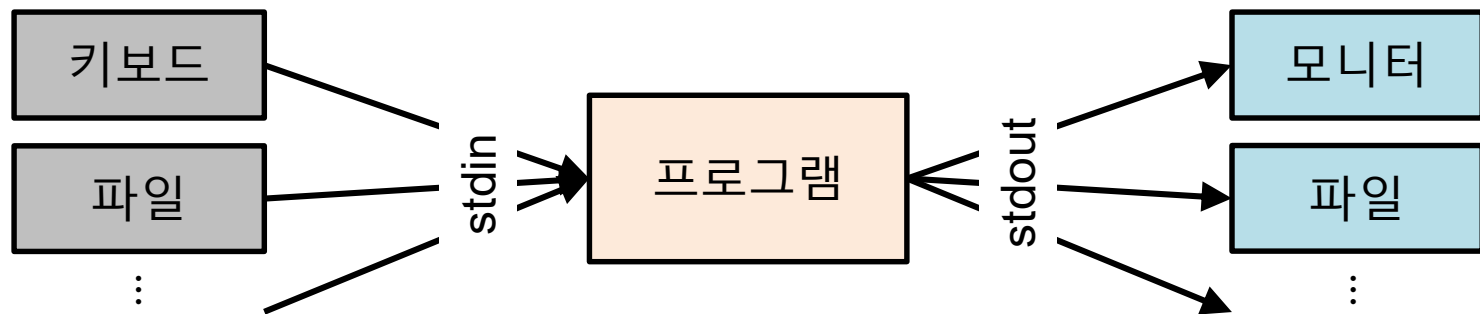
스트링

- 문자열로 저장됨
- "" 사이에 기재하지 않지만 '\0' (**null** 문자)가 문자열 끝에 들어가 있음
- \ (escape 문자)를 사용하여 특수 문자를 출력할 수 있음

| Escape 문자 | 출력 |
|----------------|------------------------------|
| \\ | \ |
| \' | ' |
| \" | " |
| \b, \t, \r, \n | backspace, tab, 처음으로 리턴, 새 행 |

콘솔 I/O

- `stdout`, `stdin`: 콘솔 출력, 입력 스트림
- `puts(string)`: `stdout`에 스트링 출력
- `putchar(char)`: `stdout`에 문자 출력
- `char = getchar()`: `stdin`로 입력된 문자를 받아들임
- `string = gets(string)`: `stdin`로 입력된 스트링을 받아들임
- ...



Preprocessor macro

- Preprocessor macro는 #로 시작됨
`#include <stdio.h>`
- `#define msg "Hello World!"`
소스 코드에서 msg는 "Hello World!"로 지정됨
- 일반적으로 `#define`을 통해 상수를 지정함
`#define PI 3.14159`
`#define EPSILON 0.00001`
`#define UNIV_NAME "KNUT"`
- 표현 macro
`#define add3 (x, y, z) ((x) + (y) + (z))`

Conditional preprocessor macro

- **#if, #ifdef, #ifndef, #else, #elif, #endif**
코드 컴파일 흐름 제어
 - 코드를 컴파일하기 전에 conditional preprocessor macro를 먼저 컴파일함
 - Conditional preprocessor macro와 사용하는 조건은 상수 또는 preprocessor define이 되어야 함
 - 다음의 gcc 옵션을 통해 preprocessor define을 지정할 수 있음
 - D name=value
 - 중복 선언 예방을 위해 헤더 파일에 많이 쓰임

Conditional preprocessor macro

- **#pragma**
preprocessor directive
- **#error**, **#warning**
사용자가 에러/경고를 지정할 수 있음
- **#undef** msg
컴파일 시 msg 선언을 제거함