

C 언어 기초 I

작성자 : 홍효상
이메일 : hyomee@naver.com
소스 : https://github.com/hyomee/c_basic

Content

I. C 언어 기초 I

1. 프로그램 기초
2. 상수, 변수, 함수
3. 라이브러리
4. 프로그램 제어

Content

I. 프로그램 기초

1. 프로그램
2. 프로그램 언어
3. C 언어
4. 개발 TOOL 설치
5. 첫번째 프로그램
6. 자료 저장 방식
7. 자료형

1. 프로그램이란

1. 프로그램 기초 1-1. 프로그램이란

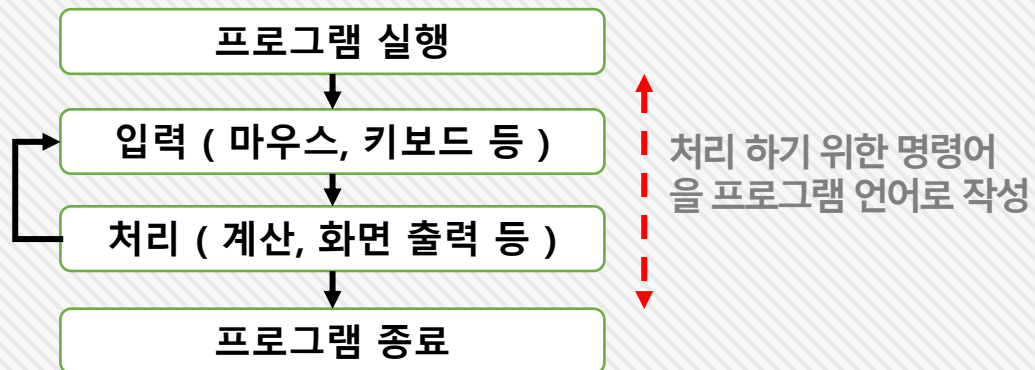
“ 어떤 목적을 달성 하기 위해서 프로그램 언어로 진행 순서를 작성 한 것 ”

사전적 의미

- 목록, 순서, 예정 계획 이란 뜻
- Rapp & Poertner(1992) - 특정 목표를 달성하기 위한 활동의 집합체
- York(1983) - 목표를 달성하기 위한 일련의 상호 의존적인 활동
- Smith(1989) - 특정 목표를 달성 하기 위해서 만들어진 조직적인 활동

프로그램 파일 (.exe)

- 어떤 목적을 수행 하기 위해 만든 파일



컴퓨터 프로그램

- 어떤 작업을 하기 위한 일련의 순서를 컴퓨터에게 알려 주기 위한 파일
- 일련의 순서를 컴퓨터가 이해 할 수 있는 명령어들의 모음
- 컴퓨터가 이해 할 수 있는 명령어는 0, 1로 되어 있는데 사람이 이해 할 수 있는 언어를 프로그램 언어라 한다.
- 두산백과 사전 : 컴퓨터를 실행 시키기 위해 차례대로 작성된 명령어 모음

프로그램 종류

- 시스템 프로그램
 - 컴퓨터 시스템과 하드웨어들을 제어 및 관리 하는 프로그램
 - 예) 윈도우, 리눅스, 장치 드라이버, 컴파일러 등
- 응용 프로그램
 - 사용자가 원하는 기능을 제공하는 프로그램
 - 엑셀, 게임, 워드 등

프로그램과 소프트웨어

- 프로그램
 - 컴파일된 결과물뿐만 아니라, 프로그래머가 작성한 소스 코드까지도 포함.
- 소프트웨어
 - 프로그램뿐만 아니라 CD, 설명서, 제품 포장 등 패키지 전체.

2. 프로그램 언어

1. 프로그램 기초 1-1. 프로그램 이란

“ 사람이 이해 할 수 있는 표현법을 사용 하여 프로그래밍 할 수 있는 언어 ”

의미

- 컴퓨터 시스템을 구동 시키는 소프트웨어를 작성하기 위한 형식언어
- 컴퓨터를 이용하여 특정 문제를 해결하기 위한 프로그램을 작성하기 위해 사용되는 언어

```
b8 21 0a 00 00
a3 0c 10 00 06
b8 6f 72 6c 64
a3 08 10 00 06
b8 6f 2c 20 57
a3 04 10 00 06
b8 48 65 6c 6c
a3 00 10 00 06
b9 00 10 00 06
ba 10 00 00 00
bb 01 00 00 00
b8 04 00 00 00
cd 80
b8 01 00 00 00
cd 80
```

```
MONITOR FOR 6802 1.4      9-14-80  TSC ASSEMBLER  PAGE    2

C000                      ORG      ROM+$0000 BEGIN MONITOR
C000 8E 00 70  START      LDS      #STACK

*****
* FUNCTION: INITA - Initialize ACIA
* INPUT: none
* OUTPUT: none
* CALLS: none
* DESTROYS: acc A

0013      RESETA  EQU      %00010011
0011      CTLREG  EQU      %00010001

C003 86 13      INITA    LDA  A  #RESETA  RESET ACIA
C005 B7 80 04          STA  A  ACIA
C008 86 11          LDA  A  #CTLREG  SET 8 BITS AND 2 STOP
C00A B7 80 04          STA  A  ACIA

C00D 7E C0 F1          JMP      SIGNON  GO TO START OF MONITOR
```

언어 종류

• 저급 언어

- 기계어

- 컴퓨터가 직접 이해할 수 있는 언어
- 0과 1의 2진수 형태로 표현되며 수행시간이 빠르다.
- CPU에 내장된 명령들을 직접 사용하는 것으로, 프로그램을 작성하고 이해하기가 어렵다.
- 기종마다 기계어가 다르므로 언어의 호환성이 없다.

- 어셈블리어

- 기계어와 1:1로 대응되는 기호로 이루어진 언어로, 니모닉(Mnemonic) 언어
- 하드웨어 제어에 주로 사용되며, 언어의 호환성이 없다.
- 컴퓨터가 직접 이해할 수 없으므로 어셈블리어로 작성된 프로그램은 어셈블러를 사용하여 기계어로 번역해주어야 한다.

• 고급 언어

- 컴파일러 언어 라고도 하며, 인간이 실생활에서 사용하는 자연어와 비슷한 형태 및 구조를 가지고 있다.
- 하드웨어에 대한 깊은 지식이 없어도 프로그램 작성과 수정이 용이
- 컴퓨터가 이해할 수 있는 기계어로 번역하기 위해 컴파일러나 인터프리터가 사용
- 기계어와 어셈블리어를 제외한 C, JAVA, Python등의 언어가 고급언어

2. 프로그램 언어

1. 프로그램 기초 1-1. 프로그램 이란

“ 사람이 이해 할 수 있는 표현법을 사용 하여 프로그래밍 할 수 있는 언어 ”

컴파일러

- 컴파일러는 고급 언어로 작성된 프로그램 전체를 목적 프로그램으로 번역한 후, 링킹 작업을 통해 컴퓨터에서 실행 가능한 실행 프로그램을 생성
- 번역 실행 과정을 거쳐야 하기 때문에 번역 과정이 번거롭고 번역 시간이 오래 걸리지만, 한번 번역한 후에는 다시 번역하지 않으므로 실행 속도가 빠르다.
- 컴파일러를 사용하는 언어에는 C언어 Java 등

컴파일러와 인터프리터 차이점

구분	컴파일러	인터프리터
번역단위	전체	행(줄)
목적 프로그램	생성함	생성하지 않음
실행속도	빠름	느림
번역속도	느림	빠름
관련언어	C, JAVA	Python, BASIC, LISP, APL, SNOBOL

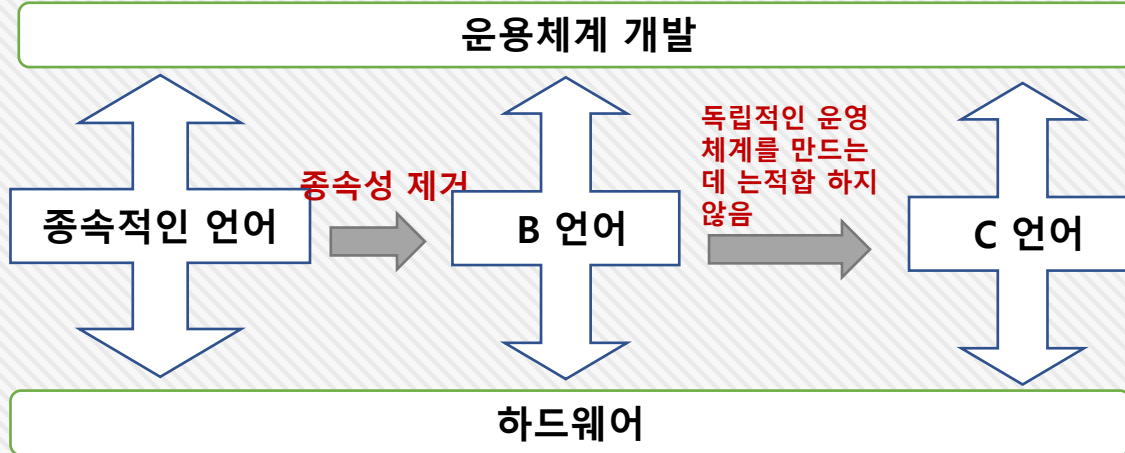
인터프리터

- 인터프리터는 고급 언어로 작성된 프로그램을 한 줄 단위로 받아들여 번역하고, 번역과 동시에 프로그램을 한 줄 단위로 즉시 실행시키는 프로그램.
- 프로그램이 직접 실행되므로 목적 프로그램은 생성되지 않음
- 줄 단위로 번역, 실행되기 때문에 시분할 시스템에 유용하며 원시 프로그램의 변화에 대한 반응이 빠르다
- 번역 속도는 빠르지만 프로그램 실행 시 매번 번역해야 하므로 실행 속도는 느리다.
- CPU의 사용시간의 낭비가 크다.
- 인터프리터를 사용하는 언어에는 Python, BASIC, SNOBOL, LISP, APL 등

“ 유닉스 운용체계를 만드는 데 사용한 언어 ”

C언어 탄생

- 1972년 켄 톰슨과 데니스 리치가 벨 연구소에서 일할 당시 새로 개발된 유닉스 운영 체제에서 사용하기 위해 개발한 프로그래밍 언어



- 하드웨어의 세밀한 부분까지 제어
- 하드웨어에 독립된 형태로 프로그램 개발
- 유닉스 시스템의 바탕 프로그램은 모두 C로 작성되었고, 수많은 운영 체제의 커널 또한 C로 만들어 졌고, 오늘날 많이 쓰이는 C++는 C에서 객체 지향형이다. => 거의 모든 커널이 C로 구현됨.
- C는 시스템 프로그램 개발에 매우 적합하지만, 응용 프로그램 개발에 사용 되기도 함.

C언어의 장단점

- 장점**
 - 다양한 하드웨어로의 이식성이 좋다.
 - 절차 지향 프로그래밍 언어로, 코드가 복잡하지 않아 상대적으로 유지 보수가 쉽다.
 - 저급 언어의 특징을 가지고 있으므로, 어셈블리어 수준으로 하드웨어를 제어할 수 있다.
 - 코드가 간결하여, 완성된 프로그램의 크기가 작고 실행 속도가 빠르다.
- 단점**
 - 저급 언어의 특징을 가지고 있으므로, 자바와 같은 다른 고급 언어보다 배우기가 쉽지 않다.
 - 다른 언어와는 달리 시스템 자원을 직접 제어할 수 있으므로, 프로그래밍하는 데 세심한 주의를 기울여야 한다.

C언어 표준

- C언어는 1989년부터 ANSI(American National Standards Institute)에서 표준화 작업을 진행 현재는 ISO/IEC에서 표준화를 담당.
- C언어의 최신 표준은 ISO/IEC사이트에서 확인 가능함

* <https://en.cppreference.com/w/> 에서 확인 가능함

4. 개발 Tool 설치

1. 프로그램 기초 1-1. 프로그램 이란

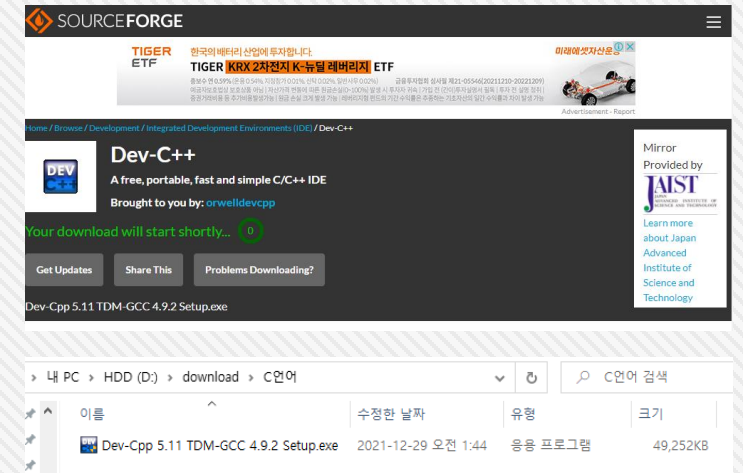
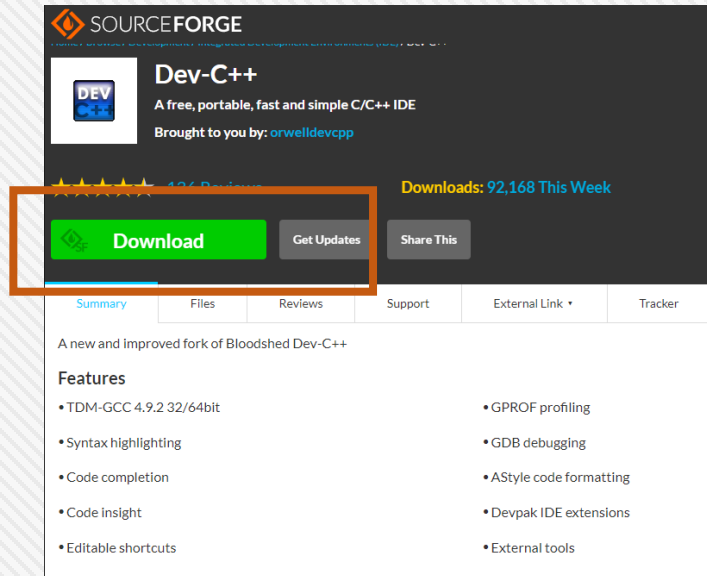
“Dev-C++은 GNU 라이선스로 자유롭게 사용가능한 C/C++ 언어 통합 개발 환경(IDE) ”

Dev-C++

- Dev-C++은 MinGW 컴파일러를 사용해서 윈도우 환경에만 실행
- MinGW는 Minimalist GNU for Windows 의 약자로 주로 32비트 오픈소스 프로젝트를 윈도우에서 돌릴 때 사용
- TDM-GCC 는 MinGW의 API를 사용해서 64비트 환경까지 사용할 수 있다
- 무료, 이동성, 빠르고 간편한 C/C++ 통합개발환경

Dev-C++ 설치 - 다운로드

- ① SourceForge 에서 다운로드 :
- <https://sourceforge.net/projects/orwelldevcpp/>
- ② 다운로드 버튼을 클릭 하고 대기 하면 다운로드 받을 폴더 선택 화면에서 다운로드 폴더 선택



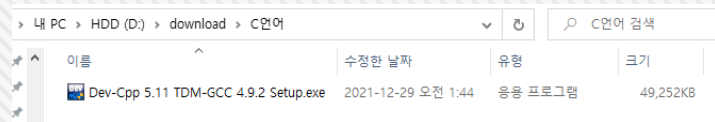
4. 개발 Tool 설치

1. 프로그램 기초 1-1. 프로그램 이란

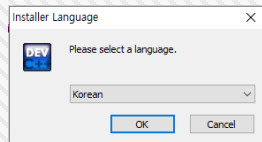
“Dev-C++은 GNU 라이선스로 자유롭게 사용가능한 C/C++ 언어 통합 개발 환경(IDE) ”

Dev-C++ 설치

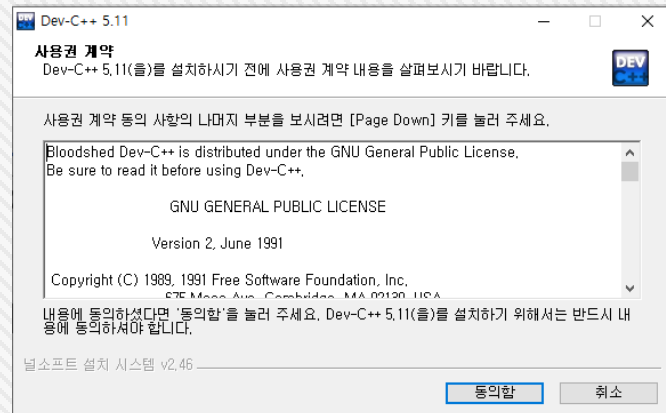
1. 다운로드 폴더에서 설치 파일을 실행 한다.



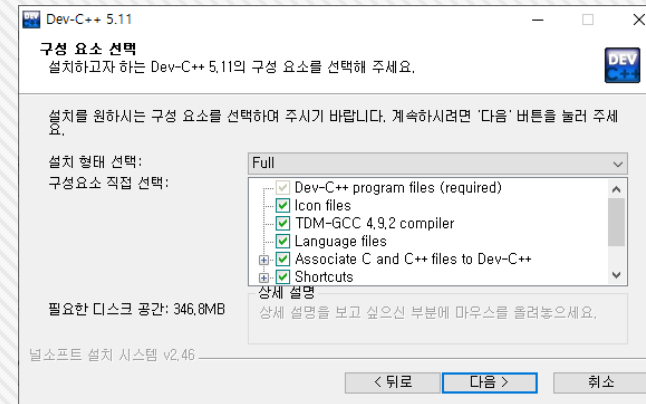
2. 언어 선택 메시지 박스에서 Korea 선택 후 OK



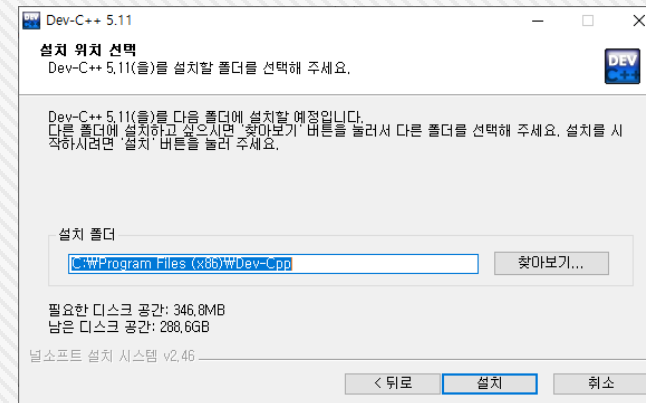
3. 사용권 계약 화면에서 “동의함” 선택



4. 구성요소 선택 화면에서 “다음” 선택 (default)



5. 설치 위치 선택 화면에 폴더 선택 후 설치 선택 (D:\Weekend\영문이름\tools)



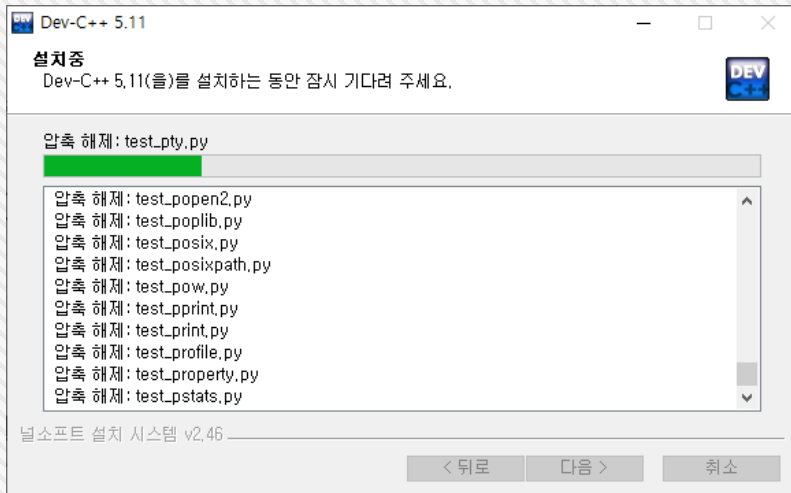
4. 개발 Tool 설치

1. 프로그램 기초 1-1. 프로그램 이란

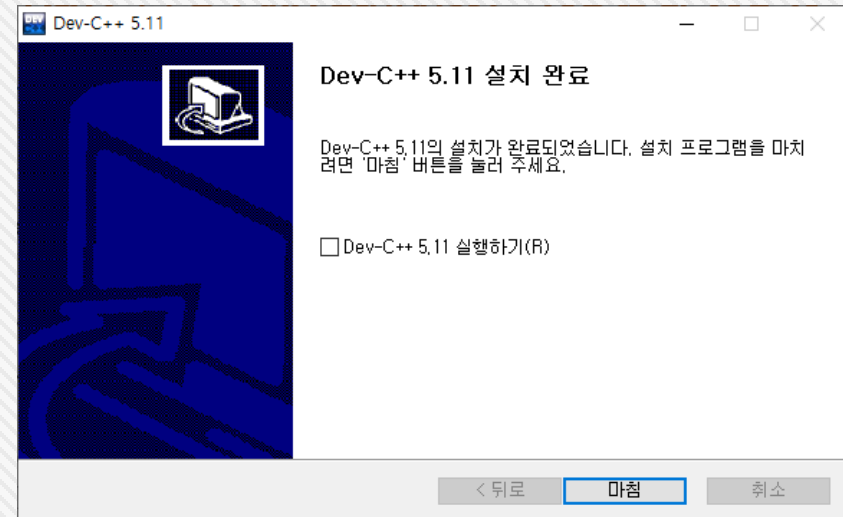
“Dev-C++은 GNU 라이선스로 자유롭게 사용가능한 C/C++ 언어 통합 개발 환경(IDE) ”

Dev-C++ 설치

5. 설치 진행 중....



6. 완료 화면에서 체크 박스 해제 후 마침 선택



4. 개발 Tool 설치 – c/c++ 컴파일러

1. 프로그램 기초
1-1. 프로그램 이란

▲ C/C++ 컴파일러 설치

1, Download : <https://sourceforge.net/projects/mingw-w64/files/mingw-w64/>

MinGW-W64 GCC-8.1.0

- [x86_64-posix-sjlj](#)
- [x86_64-posix-seh](#)
- [x86_64-win32-sjlj](#)
- [x86_64-win32-seh](#)
- [i686-posix-sjlj](#)
- [i686-posix-dwarf](#)
- [i686-win32-sjlj](#)

• [i686-win32-dwarf](#) : 32 bit 컴파일러

2. 다운로드 받은 후 압축을 풀고 d:\tools\202201\이름\c\compile\mingw64 폴더에 copy 한다.
=> 32bit : d:\tools\202201\이름\c\compile\mingw32

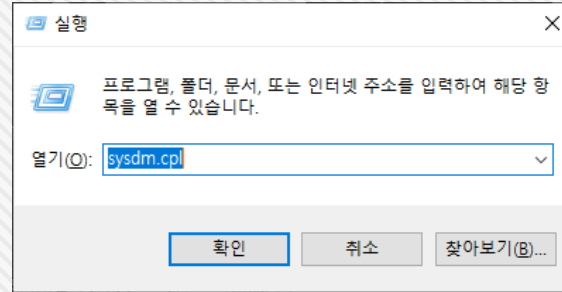
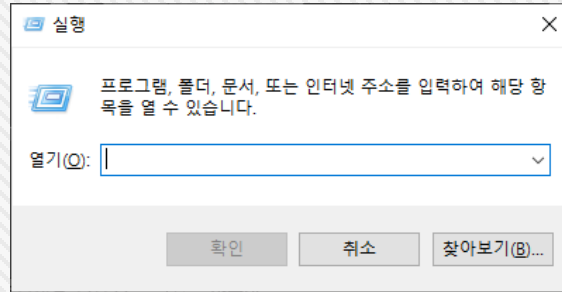
4. 개발 Tool 설치 – c/c++ 컴파일러

1. 프로그램 기초 1-1. 프로그램 이란

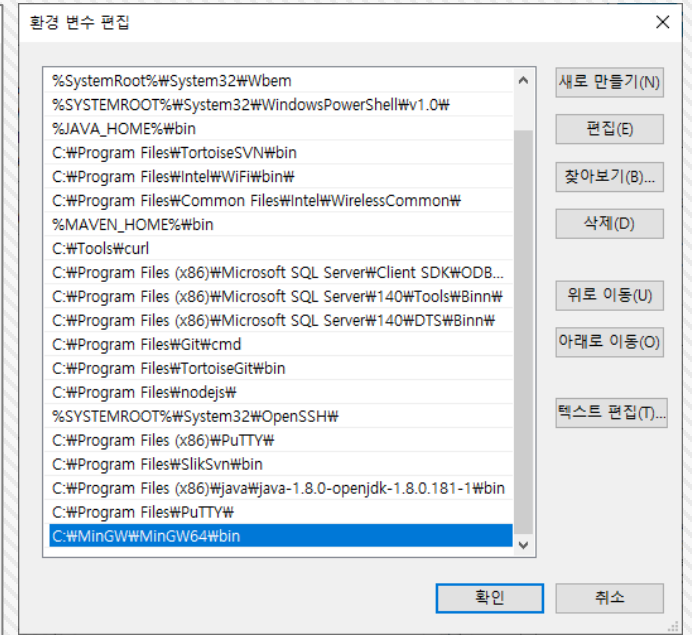
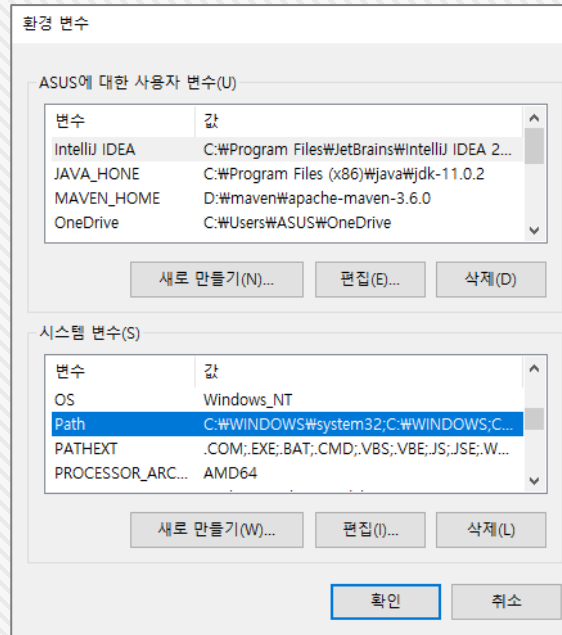
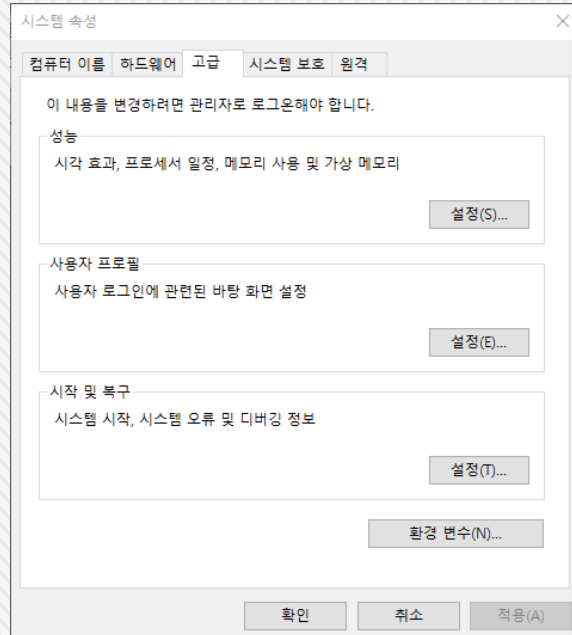
C/C++ 컴파일러 설치

3. 환경 변수 설정

- 윈도우 + R
- sysdm.cpl



4. 시스템 속성 에서 path 설정 – 새로 만들기로 2번에서 설정한 폴더 지정 ...\\mingw64\\bin



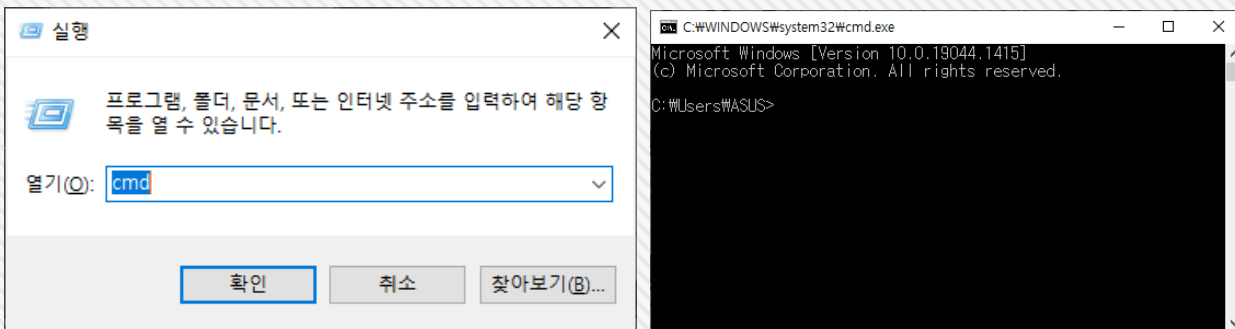
4. 개발 Tool 설치 – c/c++ 컴파일러

1. 프로그램 기초 1-1. 프로그램 이란

C/C++ 컴파일러 설치

5. cmd 실행

- 윈도우 + R
- cmd
- gcc -v



4. 개발 Tool 설치 - VSCODE

1. 프로그램 기초
1-1. 프로그램 이란

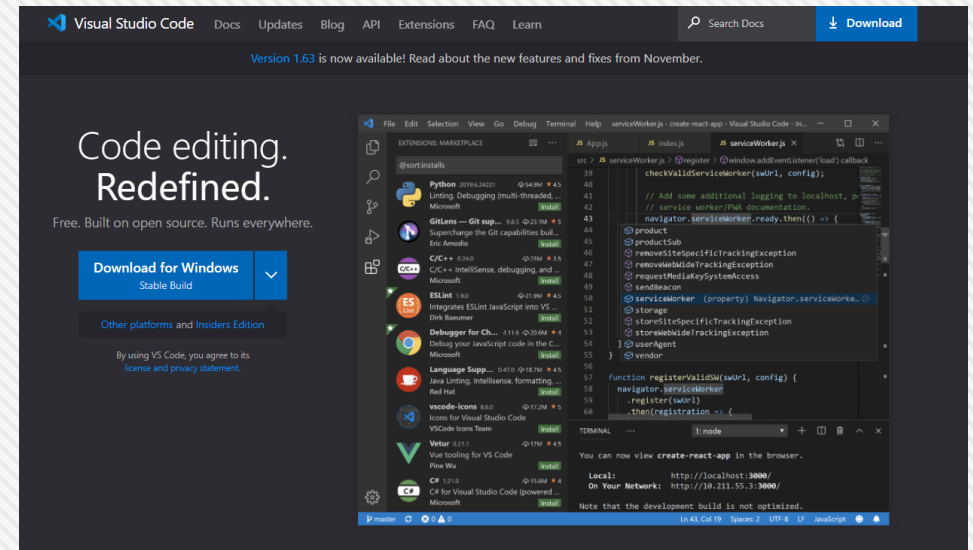
“MS의 개발 툴 중 최초로 크로스 플랫폼을 지원하는 에디터이며 윈도우, macOS, 리눅스를 모두 지원한다.”

Visual Studio Code

권장 요구 사항 (공식 문서)			
CPU		1.6 GHz 이상	
RAM		1GB 이상	
용량		500MB 이상	
운영 체제	Windows	Windows 7 이상	
	Linux	데비안 계열	우분투 16.04, 데비안 9 이상
		레드햇 계열	RHEL 7, CentOS 8, 페도라 24 이상
	macOS	10.11 El Capitan 이상	
추가 요구 사항	Windows	.NET Framework 4.5.2 이상	
	Linux	glibcxx 3.4.21, glibc 2.15 이상	

Visual Studio Code 설치 - 다운로드

- ① 다운로드 :
- <https://code.visualstudio.com/>
- ② 다운로드 버튼을 클릭 하고 대기 하면 다운로드 받을 폴더 선택 화면에서 다운로드 폴더 선택



4. 개발 Tool 설치

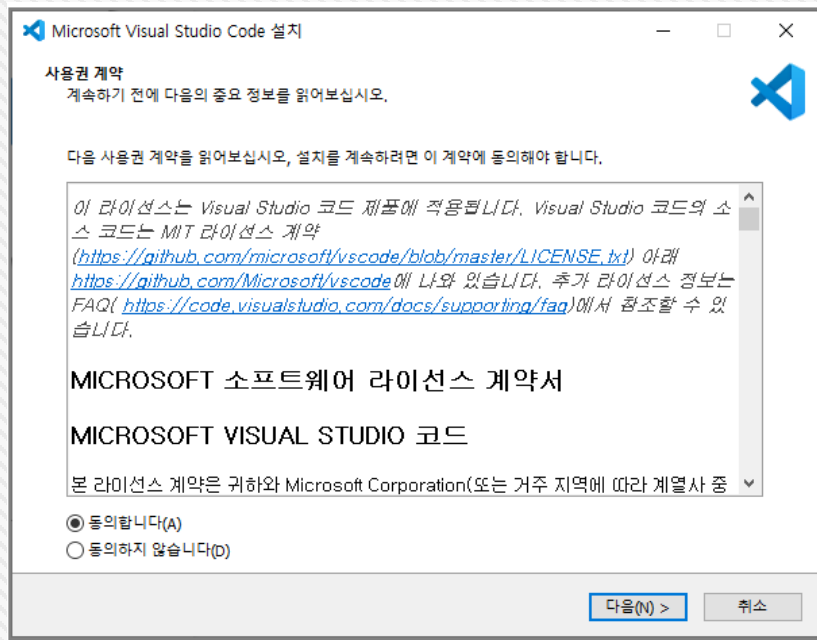
1. 프로그램 기초 1-1. 프로그램 이란

Visual Studio Code 설치

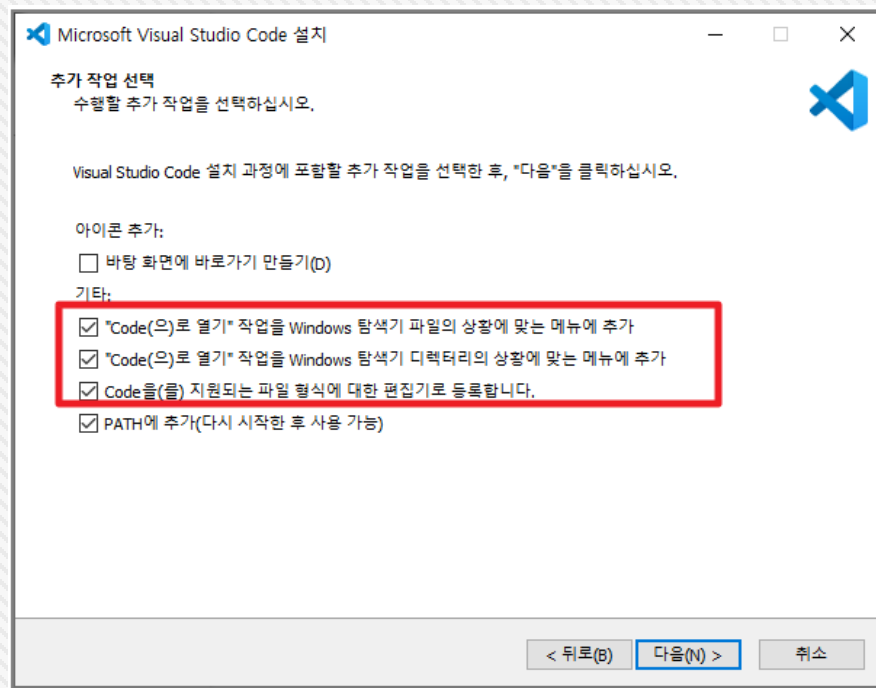
1. 다운로드 폴더에서 설치 파일을 실행 한다.

VSCodeUserSetup-x64-1.63.2.exe

2. 라이선스 동의



3. 사용상 편의를 위해 다음 3가지를 체크해준 거 외에는 별다른 옵션 변경 없이 진행했습니다.



4. 개발 Tool 설치

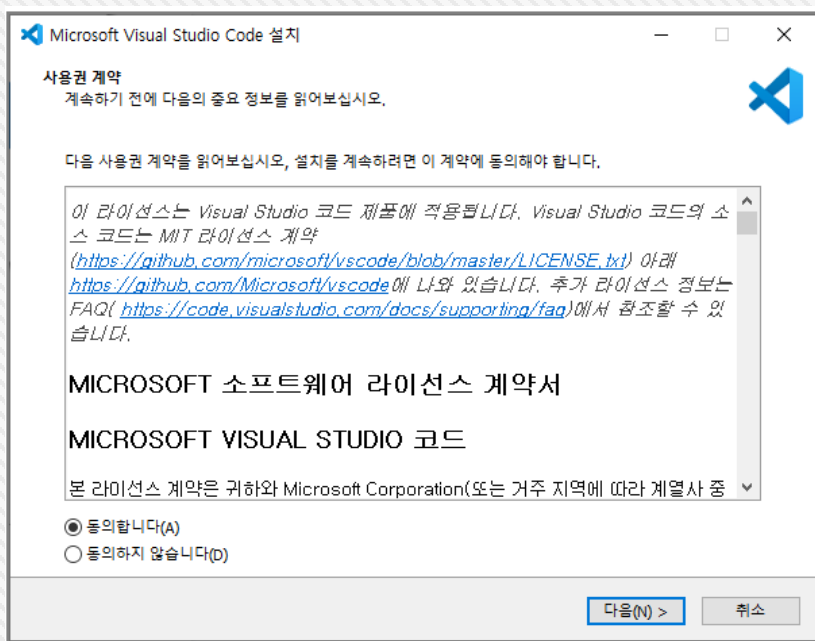
1. 프로그램 기초 1-1. 프로그램 이란

Visual Studio Code 설치

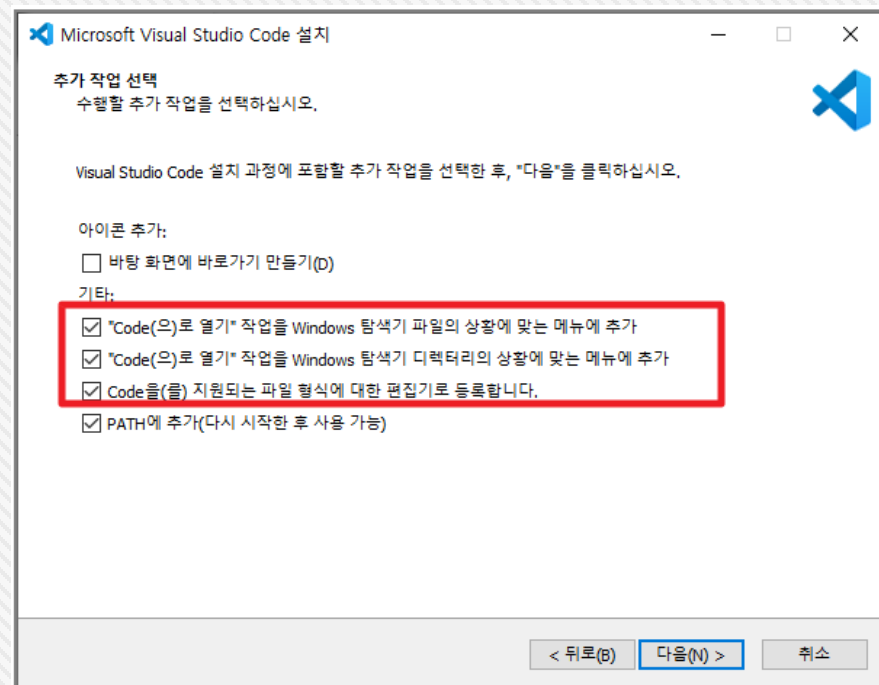
1. 다운로드 폴더에서 설치 파일을 실행 한다.

VSCodeUserSetup-x64-1.63.2.exe

2. 라이선스 동의



3. 사용상 편의를 위해 다음 3가지를 체크해준 거 외에는 별다른 옵션 변경 없이 진행했습니다.



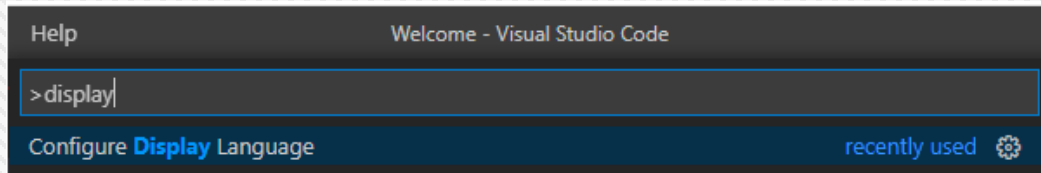
4. 개발 Tool 설치

1. 프로그램 기초 1-1. 프로그램 이란

Visual Studio Code 설치

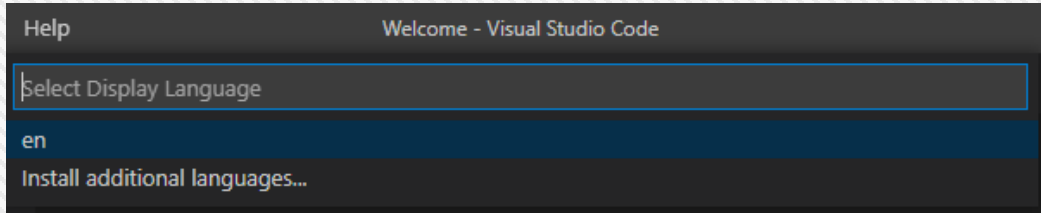
4. 언어 설정

- Ctrl + Shift + P를 누르고 입력창이 보이면 display를 입력하고 엔터



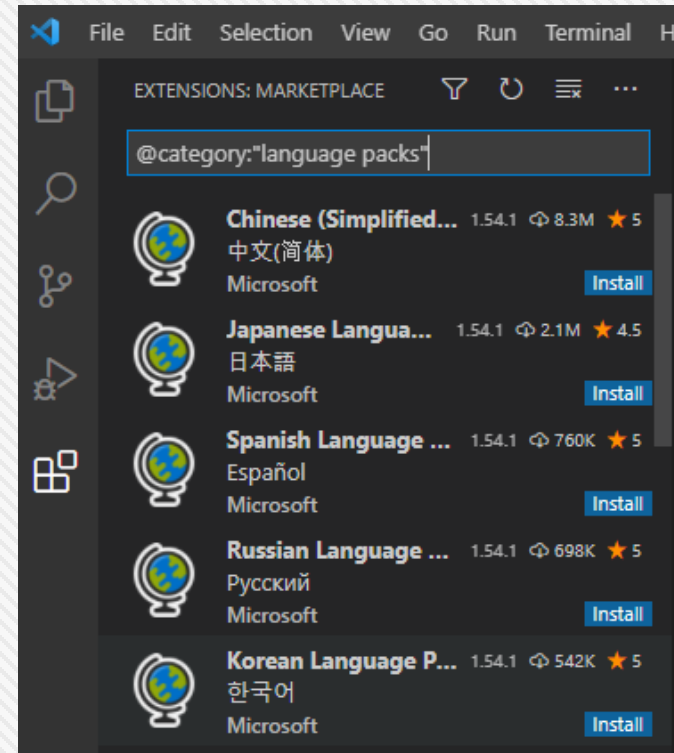
5. 언어 설정

- Install additional languages를 선택



6. 언어 설정

- Korean Language Pack을 옆에 보이는 파란색 Install 버튼을 클릭하여 설치

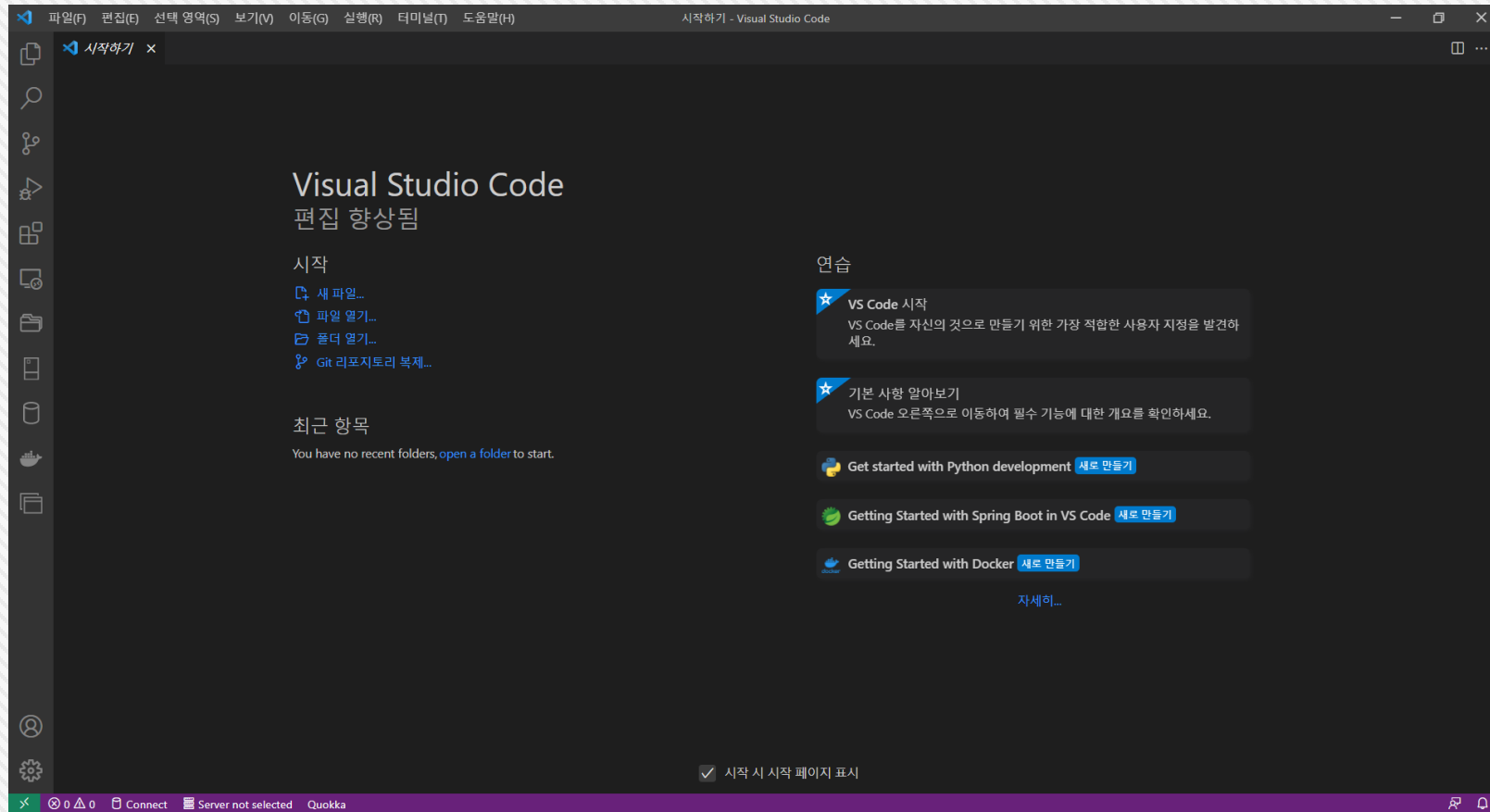


4. 개발 Tool 설치

1. 프로그램 기초 1-1. 프로그램 이란

Visual Studio Code 설치

7. 설치 완료후 Visual Studio Code를 종료했다가 다시 시작하면 유저 인터페이스가 한글로 변경

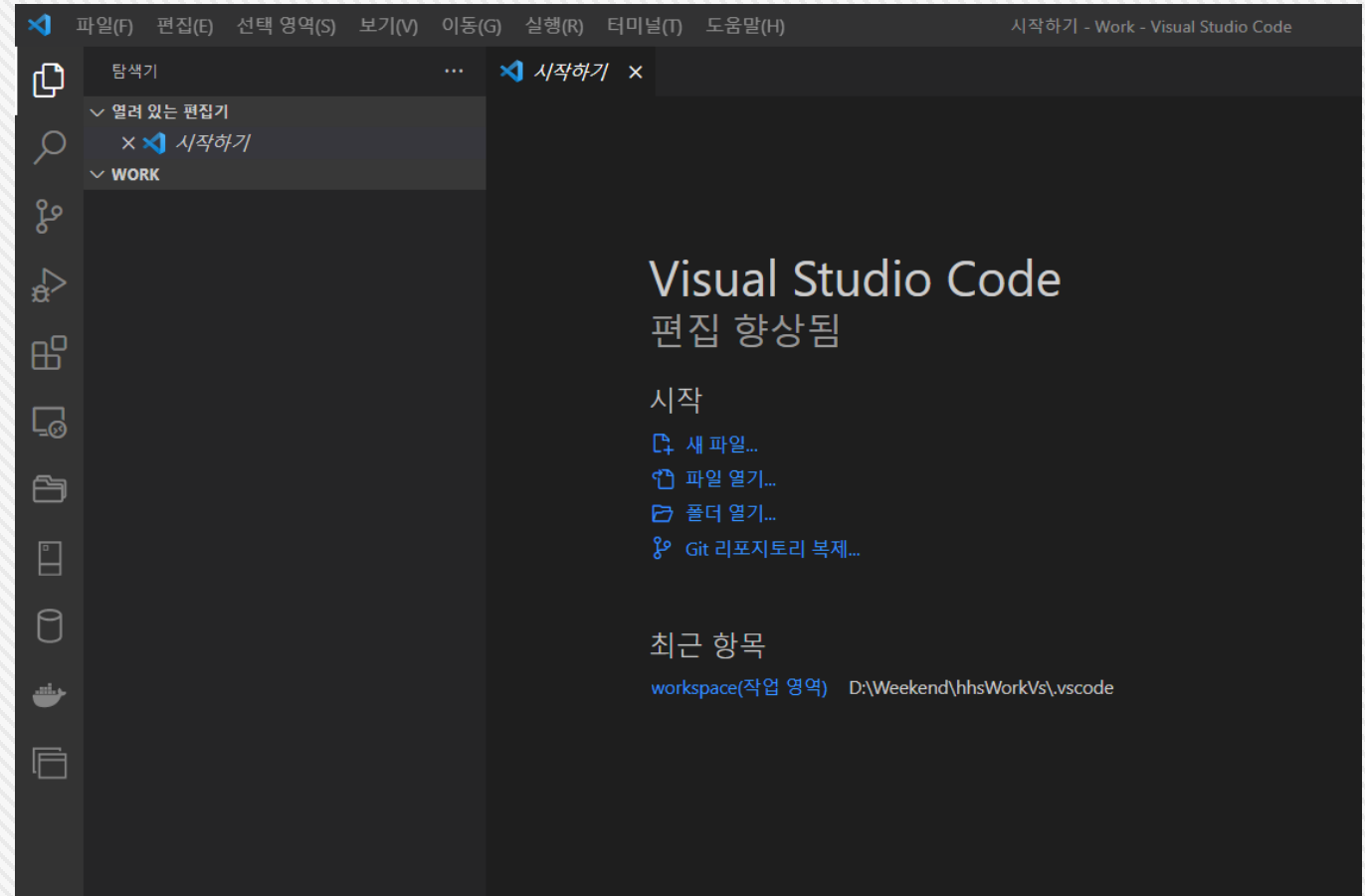
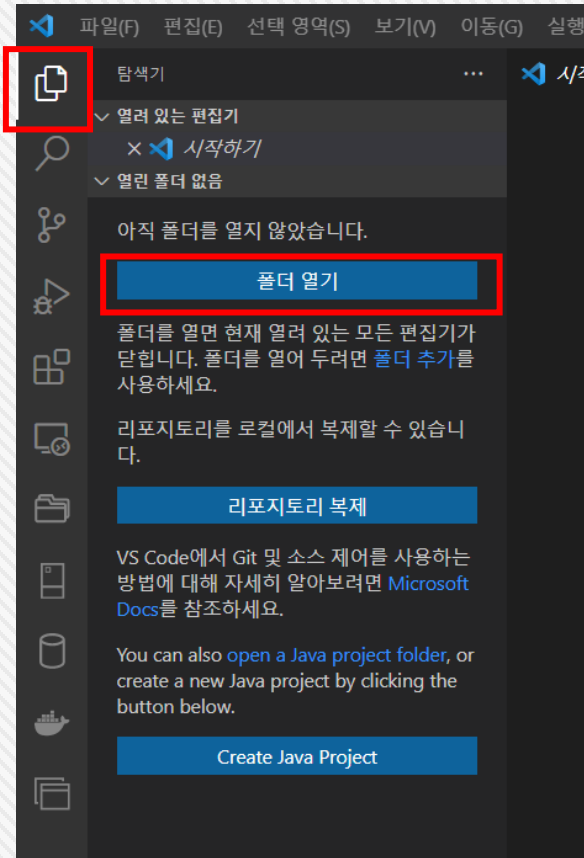


4. 개발 Tool 설치

1. 프로그램 기초 1-1. 프로그램 이란

Visual Studio Code 설치

8. 액티비티 바에서 탐색기 아이콘을 클릭하거나, 단축키 Ctrl + Shift + E를 누르면 아래 캡처화면처럼 사이드바에 탐색기 오픈

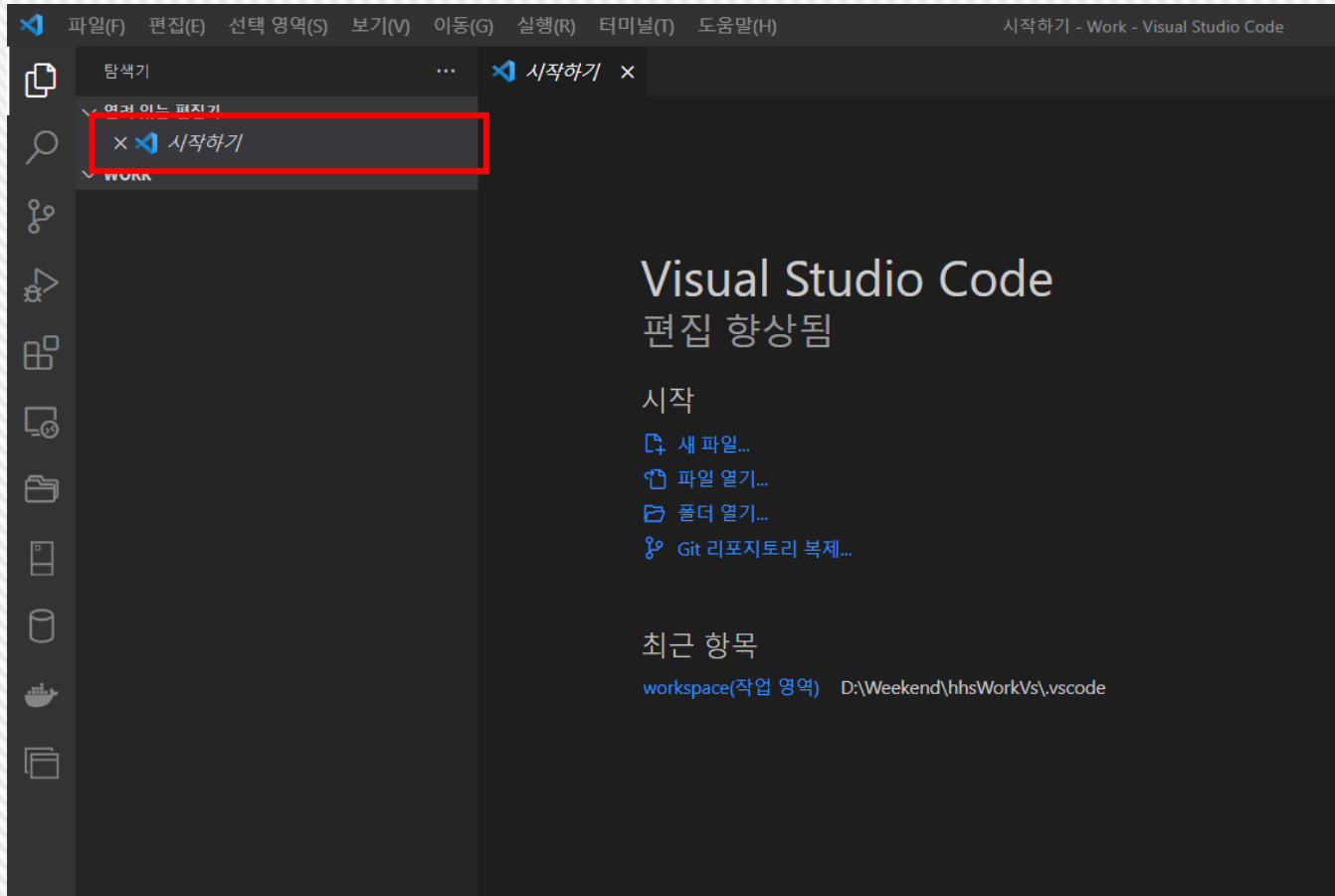


4. 개발 Tool 설치

1. 프로그램 기초 1-1. 프로그램 이란

Visual Studio Code 설치

9. 시작하기 선택



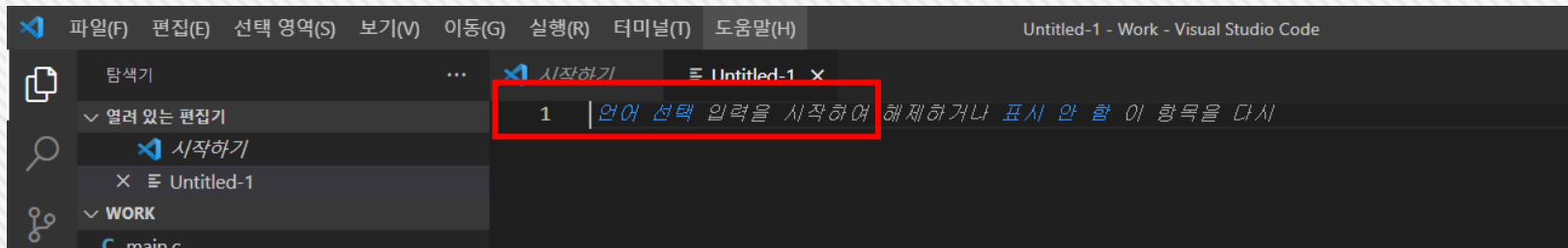
4. 개발 Tool 설치

1. 프로그램 기초
1-1. 프로그램 이란

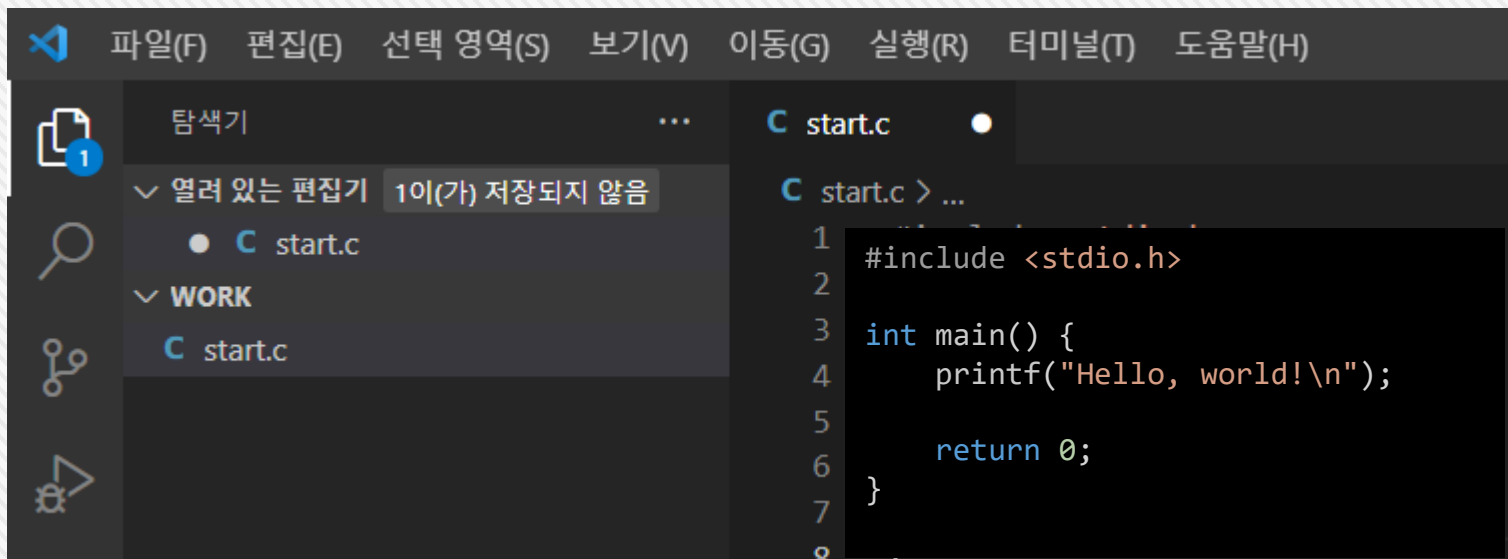
“Dev-C++은 GNU 라이선스로 자유롭게 사용가능한 C/C++ 언어 통합 개발 환경(IDE) ”

Visual Studio Code 설치

10. 언어 선택 하기 – C언어 선택



11. 다음을 입력 하고 저장 start.c로 저장 한다.

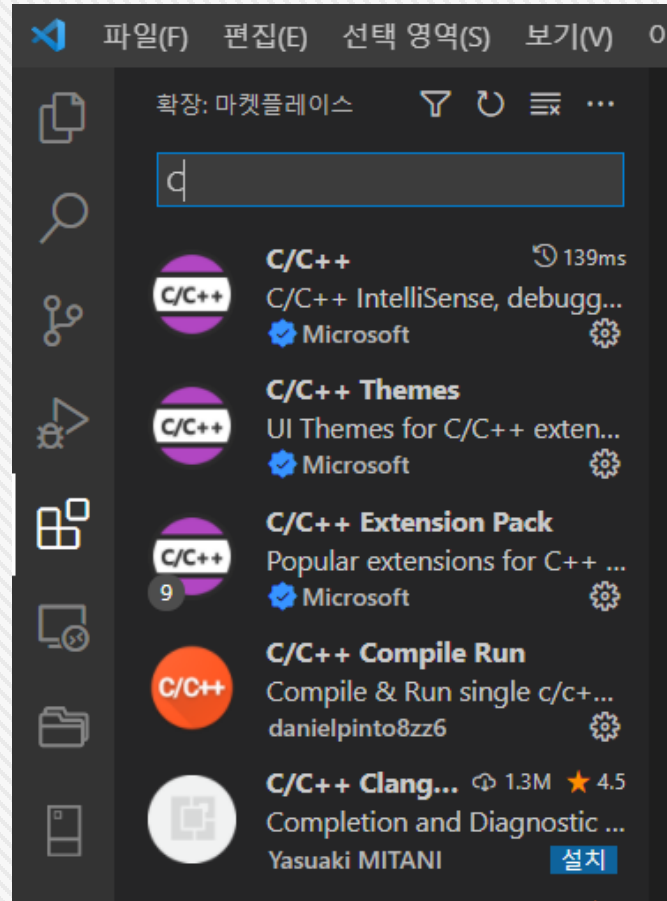


4. 개발 Tool 설치

1. 프로그램 기초 1-1. 프로그램 이란

Visual Studio Code 설치

12. 확장 팩 설치



4. 개발 Tool 설치

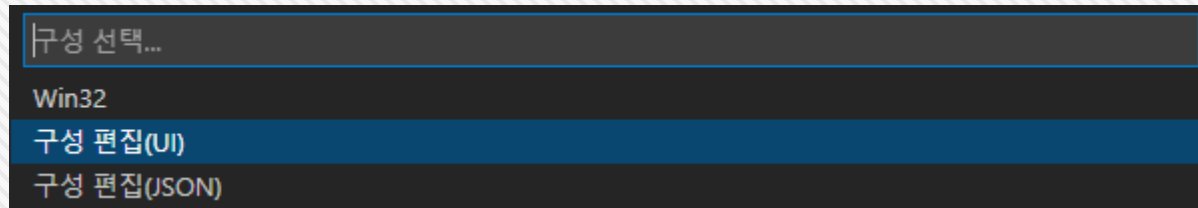
1. 프로그램 기초 1-1. 프로그램 이란

Visual Studio Code 설치

13. Ctrl + Shift + P를 눌러서 보이는 입력 창에 c/c++을 입력한 후, "C/C++: 구성 선택.."을 선택



14. 구성 편집(UI)를 선택



4. 개발 Tool 설치

1. 프로그램 기초 1-1. 프로그램 이란

Visual Studio Code 설치

15. IntelliSense 구성 에서 다음과 같이 설정 한다.

구성 이름

구성을 식별하는 이름입니다. Linux, Mac 및 Win32은(는) 해당 플랫폼에서 자동으로 선택되는 구성의 특수 식별자입니다.

편집할 구성 세트를 선택합니다.

Win32

구성 추가

컴파일러 경로

더 정확한 IntelliSense를 사용하도록 설정하는 데 사용되는, 프로젝트를 빌드하는 데 사용하는 컴파일러의 전체 경로입니다(예: /usr/bin/gcc). 확장에서는 컴파일러에 쿼리하여 IntelliSense에 사용할 시스템 포함 경로 및 기본 정의를 확인합니다.

컴파일러 경로를 지정하거나 드롭다운 목록에서 검색된 컴파일러 경로를 선택합니다.

C:\MinGW\MinGW64\bin\gcc.exe

IntelliSense 모드

MSVC, gcc 또는 Clang의 플랫폼 및 아키텍처 변형에 매핑되는 사용할 IntelliSense 모드입니다. 설정되지 않거나 \${default}(으)로 설정된 경우 확장에서 해당 플랫폼의 기본값을 선택합니다. Windows의 경우 기본값인 windows-msvc-x64(으)로 설정되고, Linux의 경우 기본값인 linux-gcc-x64(으)로 설정되며, macOS의 경우 기본값인 macos-clang-x64(으)로 설정됩니다. \${default} 모드를 재정의하려면 특정 IntelliSense 모드를 선택합니다. <compiler>-<architecture> 변형(예: gcc-x64)만 지정하는 IntelliSense 모드는 레거시 모드이며 호스트 플랫폼에 따라 <platform>-<compiler>-<architecture> 변형으로 자동으로 변환됩니다.

windows-gcc-x64

경로 포함

포함 경로는 소스 파일에 포함된 헤더 파일(예: #include "myHeaderFile.h")을 포함하는 폴더입니다. 포함된 헤더 파일을 검색하는 동안 사용할 IntelliSense 엔진의 경로 목록을 지정합니다. 이러한 경로 검색은 비재귀적입니다. 재귀적 검색을 나타내려면 **을(를) 지정합니다. 예를 들어 \${workspaceFolder}/**은(는) 모든 하위 디렉터리를 검색하지만 \${workspaceFolder}은(는) 그렇지 않습니다. Visual Studio가 설치된 Windows를 사용하거나 compilerPath 설정에 컴파일러가 지정된 경우 이 목록에 시스템 포함 경로를 나열할 필요가 없습니다.

줄당 하나의 포함 경로입니다.

\${workspaceFolder}/**

정의

파일을 구문 분석하는 동안 사용할 IntelliSense 엔진의 전처리기 정의 목록입니다. 필요에 따라 =을(를) 사용하여 값(예: VERSION=1)을 설정할 수 있습니다.

줄당 하나의 정의입니다.

_DEBUG
_UNICODE
_UNICODE

C 표준

IntelliSense에 사용할 C 언어 표준의 버전입니다. 참고: GNU 표준은 GNU 정의를 가져오기 위해 설정된 컴파일러를 쿼리하는 데만 사용되며, IntelliSense는 해당 C 표준 버전을 에뮬레이트합니다.

c17

C++ 표준

IntelliSense에 사용할 C++ 언어 표준의 버전입니다. 참고: GNU 표준은 GNU 정의를 가져오기 위해 설정된 컴파일러를 쿼리하는 데만 사용되며, IntelliSense는 해당 C++ 표준 버전을 에뮬레이트합니다.

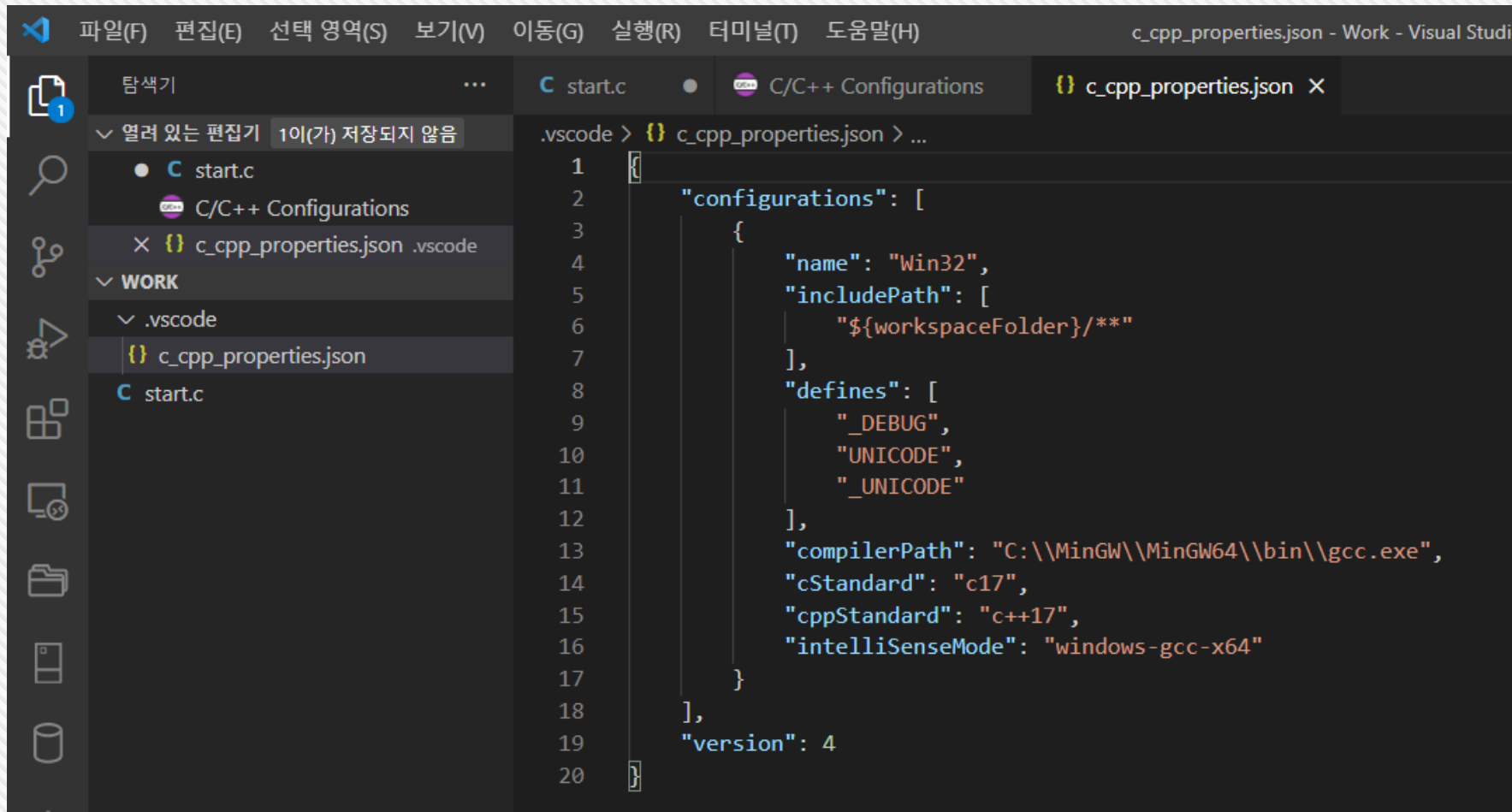
c++17

4. 개발 Tool 설치

1. 프로그램 기초 1-1. 프로그램 이란

Visual Studio Code 설치

15. IntelliSense 구성 정보 생성 됨 .



The screenshot shows the Visual Studio Code interface with the `c_cpp_properties.json` file open. The file contains the following configuration:

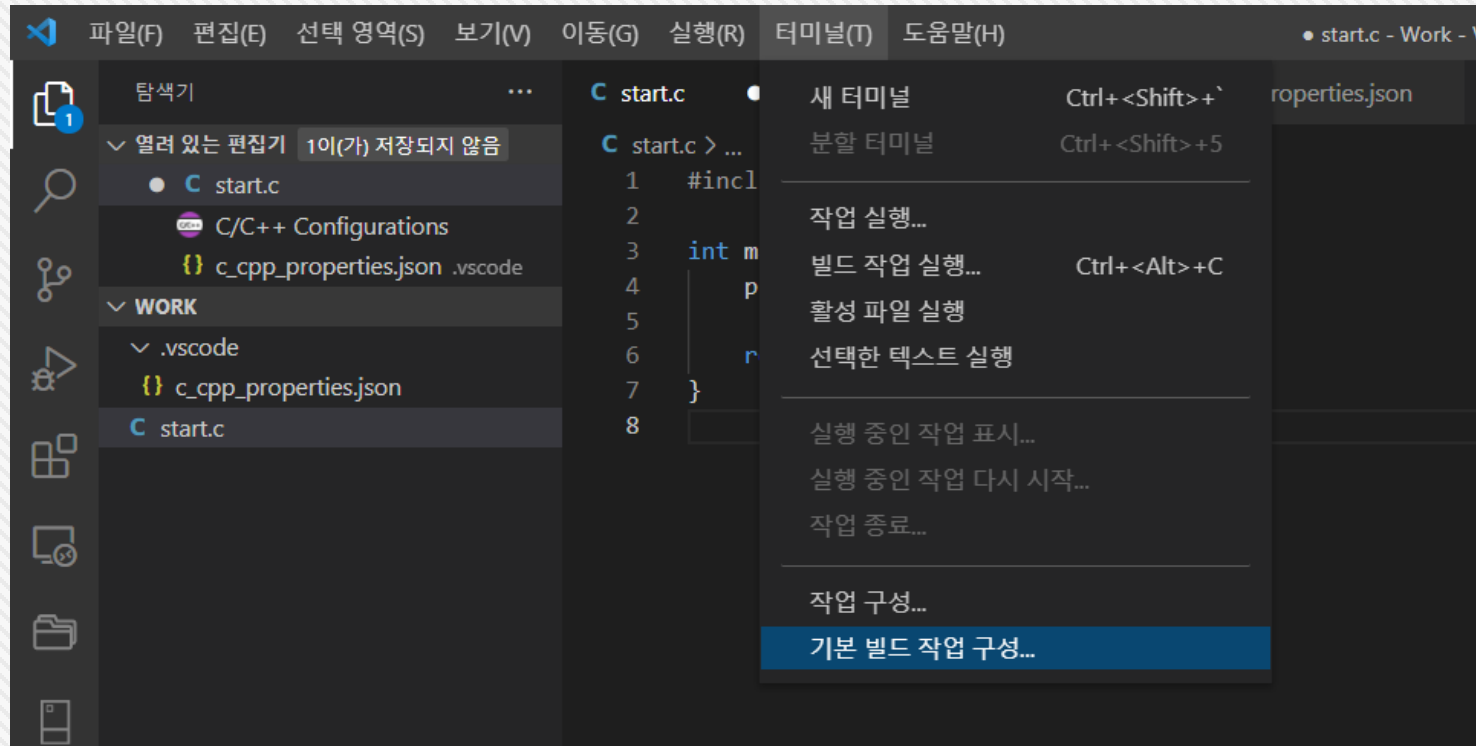
```
1  {}  
2  "configurations": [  
3    {  
4      "name": "Win32",  
5      "includePath": [  
6        "${workspaceFolder}/**"  
7      ],  
8      "defines": [  
9        "_DEBUG",  
10       "UNICODE",  
11       "_UNICODE"  
12     ],  
13     "compilerPath": "C:\\MinGW\\MinGW64\\bin\\gcc.exe",  
14     "cStandard": "c17",  
15     "cppStandard": "c++17",  
16     "intelliSenseMode": "windows-gcc-x64"  
17   }  
18 ],  
19 "version": 4  
20 }
```

4. 개발 Tool 설치

1. 프로그램 기초 1-1. 프로그램 이란

Visual Studio Code 설치

16. 코드 컴파일 및 실행 환경 설정 .



기본 빌드 작업으로 사용할 작업을 선택

C/C++: gcc.exe 활성 파일 빌드

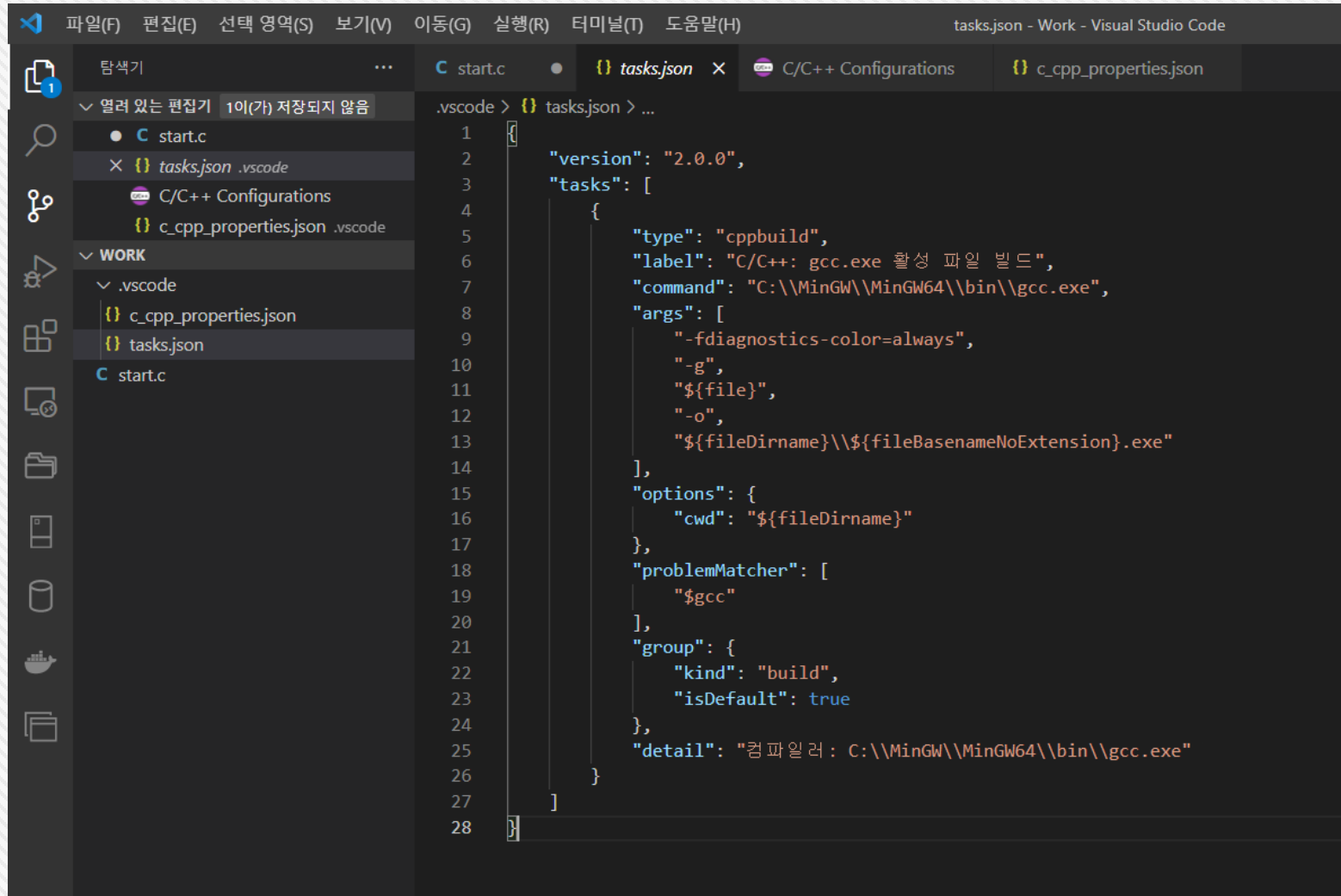
컴파일러: C:\MinGW\MinGW64\bin\gcc.exe

4. 개발 Tool 설치

1. 프로그램 기초 1-1. 프로그램 이란

Visual Studio Code 설치

17. task.json.



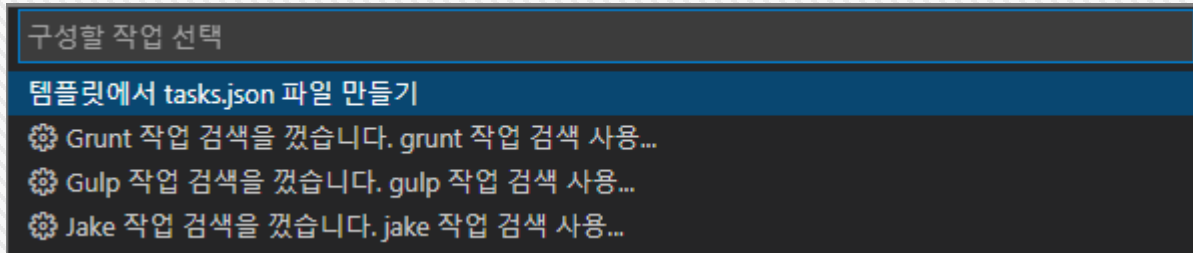
```
.vscode > {} tasks.json > ...
1  {}
2  "version": "2.0.0",
3  "tasks": [
4      {
5          "type": "cppbuild",
6          "label": "C/C++: gcc.exe 활성 파일 빌드",
7          "command": "C:\\MinGW\\MinGW64\\bin\\gcc.exe",
8          "args": [
9              "-fdiagnostics-color=always",
10             "-g",
11             "${file}",
12             "-o",
13             "${fileDirname}\\${fileBasenameNoExtension}.exe"
14         ],
15         "options": {
16             "cwd": "${fileDirname}"
17         },
18         "problemMatcher": [
19             "$gcc"
20         ],
21         "group": {
22             "kind": "build",
23             "isDefault": true
24         },
25         "detail": "컴파일러 : C:\\MinGW\\MinGW64\\bin\\gcc.exe"
26     }
27 ]
28 }
```

4. 개발 Tool 설치

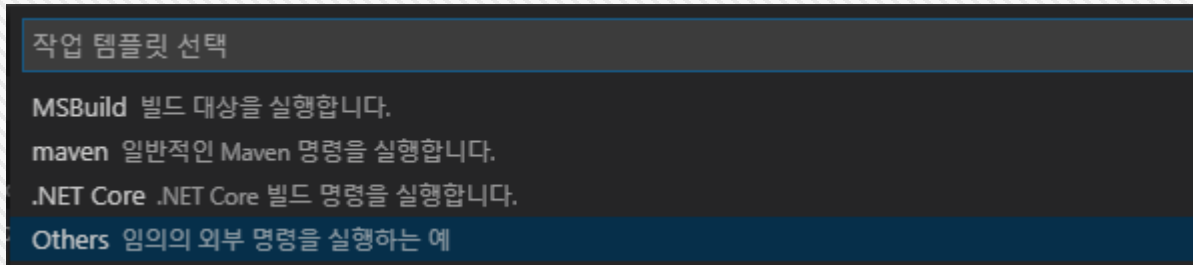
1. 프로그램 기초 1-1. 프로그램 이란

Visual Studio Code 설치

16-1. 코드 컴파일 및 실행 환경 설정 - 16 메뉴가 보이지 않는 경우.



16-2. Others를 선택

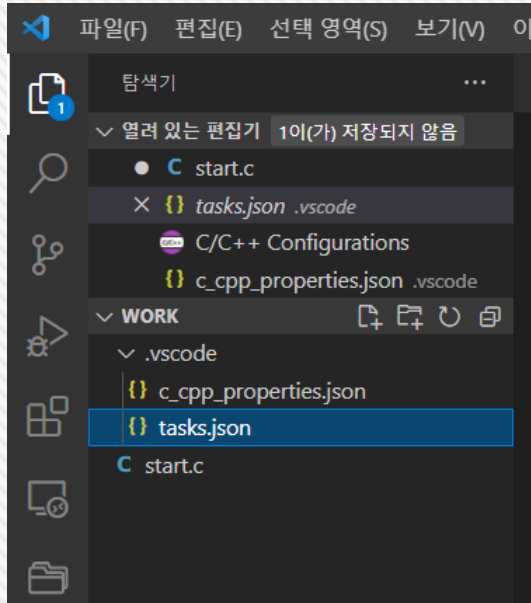


4. 개발 Tool 설치

1. 프로그램 기초 1-1. 프로그램 이란

Visual Studio Code 설치

16-3. 탐색기의 .vscode 폴더에 tasks.json 파일이 추가되고 편집기에서 해당 파일을 엽니다.



```
{
  "version": "2.0.0",
  "tasks": [
    {
      "type": "cppbuild",
      "label": "C/C++: gcc.exe 활성 파일 빌드",
      "command": "C:\\MinGW\\MinGW64\\bin\\gcc.exe",
      "args": [
        "-fdiagnostics-color=always",
        "-g",
        "${file}",
        "-o",
        "${fileDirname}\\${fileBasenameNoExtension}.exe"
      ],
      "options": {
        "cwd": "${fileDirname}"
      },
      "problemMatcher": [
        "$gcc"
      ],
      "group": {
        "kind": "build",
        "isDefault": true
      },
      "detail": "컴파일러: C:\\MinGW\\MinGW64\\bin\\gcc.exe"
    }
  ]
}
```

4. 개발 Tool 설치

1. 프로그램 기초 1-1. 프로그램 이란

Visual Studio Code 설치

15. tasks.json을 다음과 같이 수정 한다..

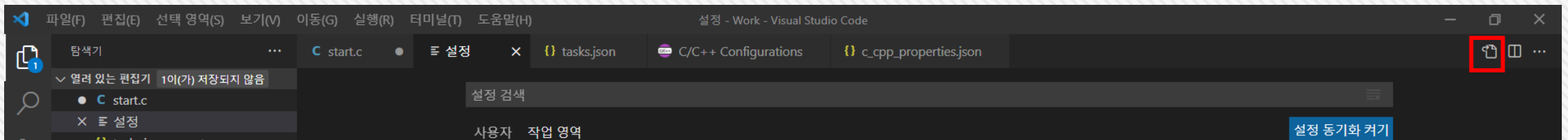
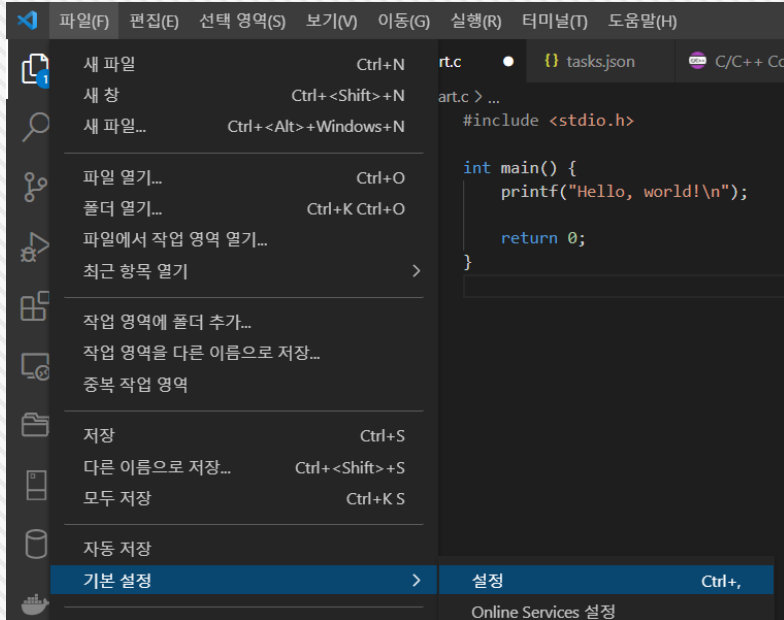
```
{
  "version": "2.0.0",
  "runner": "terminal",
  "type": "shell",
  "echoCommand": true,
  "presentation": { "reveal": "always" },
  "tasks": [
    // C 컴파일
    {
      "label": "save and compile for C",
      "command": "gcc",
      "args": [
        "${file}",
        "-g",
        "-o",
        "${fileDirname}/${fileBasenameNoExtension}"
      ],
      "group": "build",
      "problemMatcher": {
        "fileLocation": [
          "relative",
          "${workspaceRoot}"
        ],
        "pattern": {
          "regexp": "^(.*):(\\d+):(\\d+):\\s+(warning error):\\s+(.*)$",
          "file": 1,
          "line": 2,
          "column": 3,
          "severity": 4,
          "message": 5
        }
      }
    },
    // 바이너리 실행(Windows)
    {
      "label": "execute",
      "command": "cmd",
      "group": "test",
      "args": [
        "/C", "${fileDirname}\\${fileBasenameNoExtension}"
      ]
    }
  ]
}
```

4. 개발 Tool 설치

1. 프로그램 기초 1-1. 프로그램 이란

Visual Studio Code 설치

18. 단축키 설정

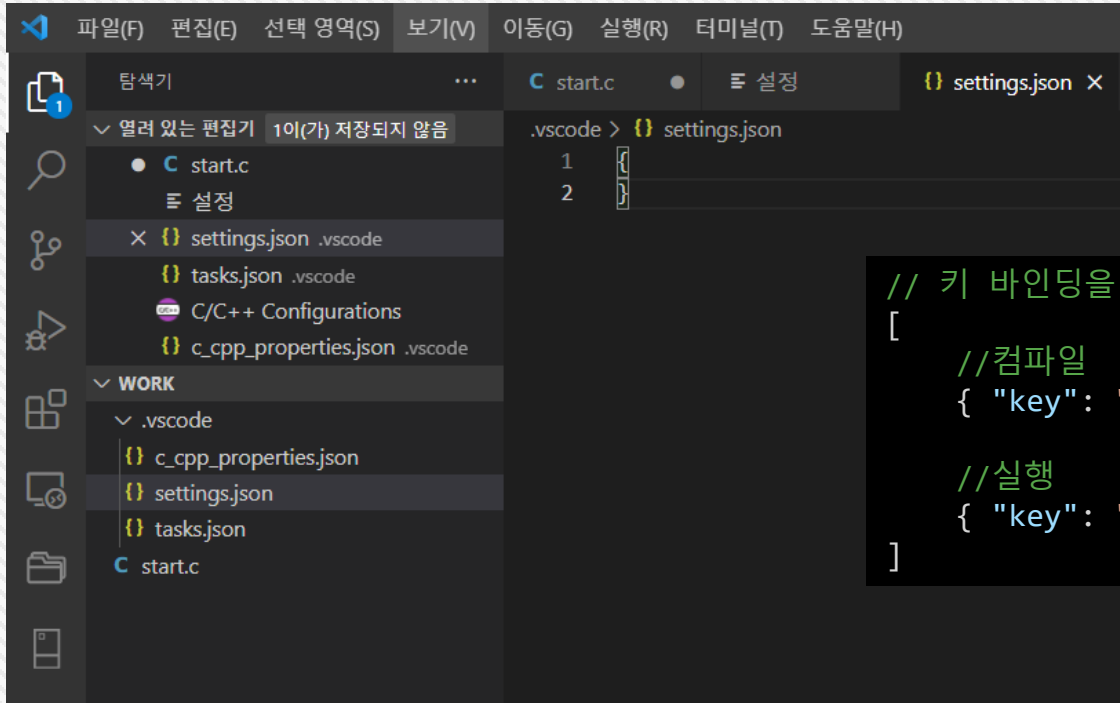


4. 개발 Tool 설치

1. 프로그램 기초 1-1. 프로그램 이란

Visual Studio Code 설치

18. settings.json



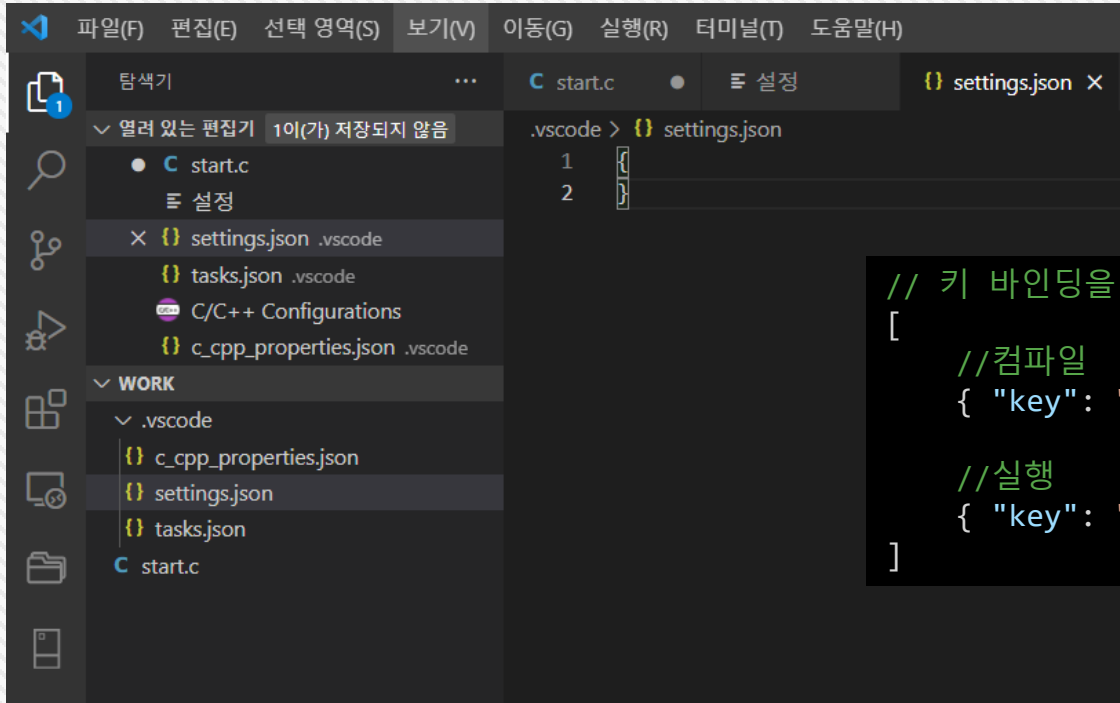
```
// 키 바인딩을 이 파일에 넣어서 기본값을 덮어씁니다.  
[  
  //컴파일  
  { "key": "ctrl+alt+c", "command": "workbench.action.tasks.build" },  
  
  //실행  
  { "key": "ctrl+alt+r", "command": "workbench.action.tasks.test" }  
]
```


4. 개발 Tool 설치

1. 프로그램 기초 1-1. 프로그램 이란

Visual Studio Code 설치

18. settings.json



```
// 키 바인딩을 이 파일에 넣어서 기본값을 덮어씁니다.  
[  
  //컴파일  
  { "key": "ctrl+alt+c", "command": "workbench.action.tasks.build" },  
  
  //실행  
  { "key": "ctrl+alt+r", "command": "workbench.action.tasks.test" }  
]
```

4. 개발 Tool 설치

1. 프로그램 기초 1-1. 프로그램 이란

Visual Studio Code 설치

19. Ctrl+alt+c 커파일

```
터미널 문제 출력 디버그 콘솔

> Executing task: C/C++: gcc.exe 활성 파일 빌드 <

빌드를 시작하는 중...
C:\MinGW\MinGW64\bin\gcc.exe -fdiagnostics-color=always -g D:\Work\start.c -o D:\Work\start.exe
빌드가 완료되었습니다.

터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.
```

20. Ctrl+alt+r 실행

```
실행할 테스트 작업 선택

실행할 테스트 작업이 없습니다. 작업 구성...
```

```
구성할 작업 선택

C/C++: gcc.exe 활성 파일 빌드
컴파일러: C:\MinGW\MinGW64\bin\gcc.exe
C/C++: gcc.exe 활성 파일 빌드
컴파일러: C:\MinGW\MinGW64\bin\gcc.exe
⚙ Grunt 작업을 검색했습니다. grunt 작업 검색 사용...
⚙ Gulp 작업을 검색했습니다. gulp 작업 검색 사용...
⚙ Jake 작업을 검색했습니다. jake 작업 검색 사용...
```

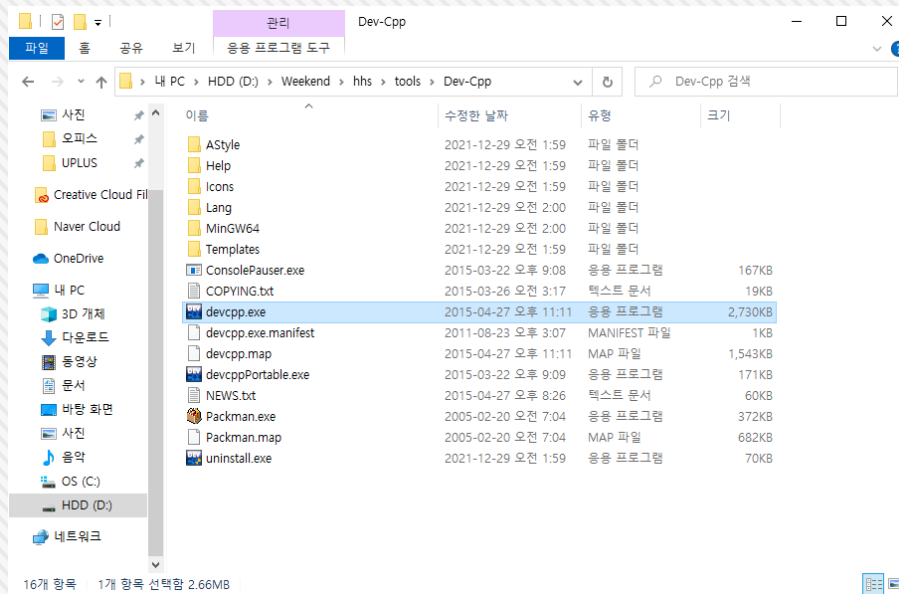
5. 첫번째 프로그램

1. 프로그램 기초 1-1. 프로그램 이란

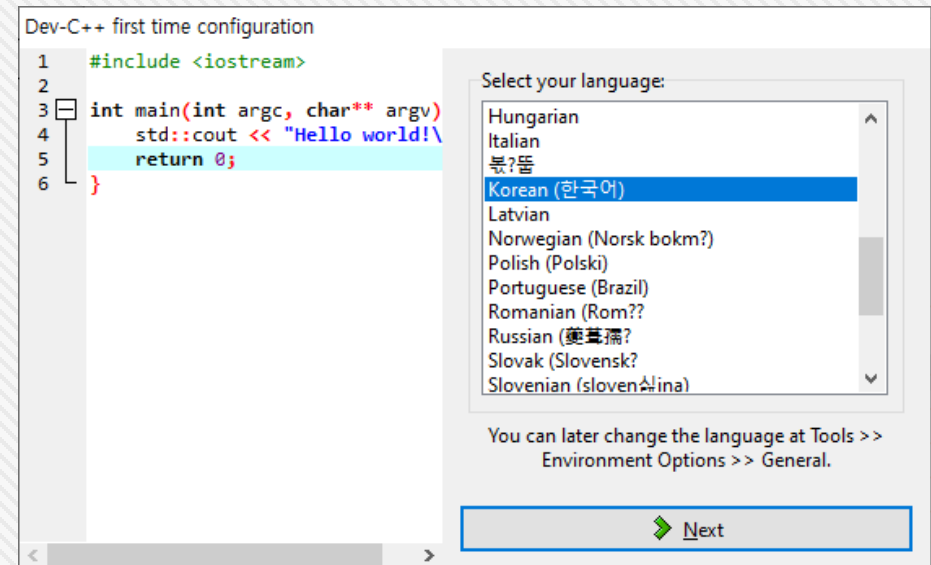
“ Hello C 언어”

Dev-C ++ 실행 후 최초 환경 설정

1. Dev C++ 실행 (설치 폴더로 이동)
: devcpp.exe



2. 환경 설정 화면에서 Korea(한국어) 선택 후 Next 클릭



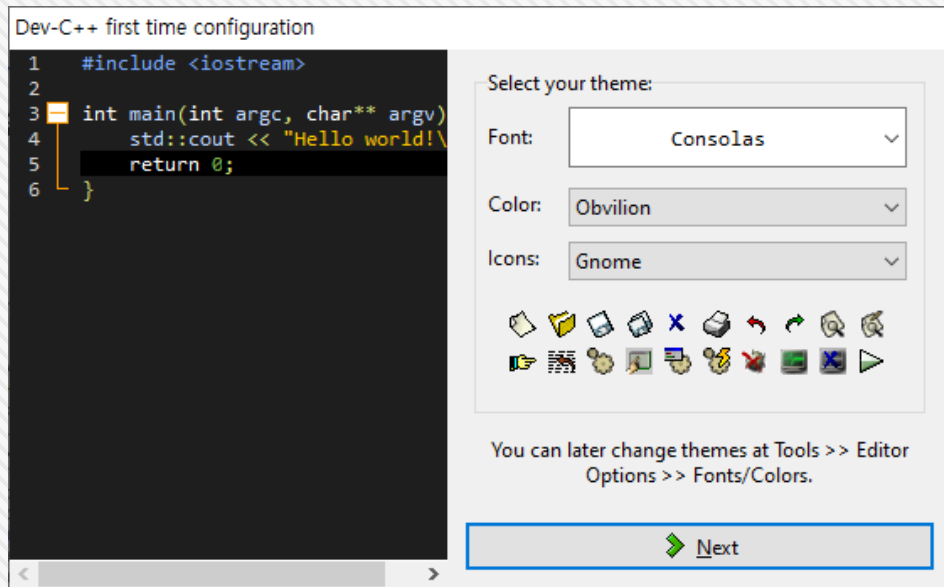
5. 첫번째 프로그램

1. 프로그램 기초 1-1. 프로그램 이란

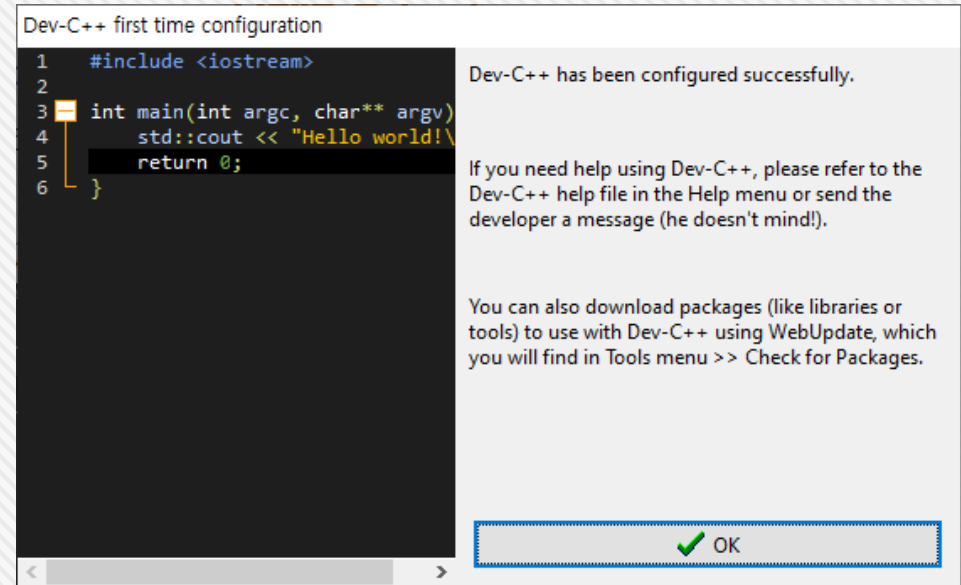
“ Hello C 언어”

Dev-C++ 실행 후 최초 환경 설정

3. 원하는 테마 선택 후 Next 클릭



4. 설정 완료 ok 클릭



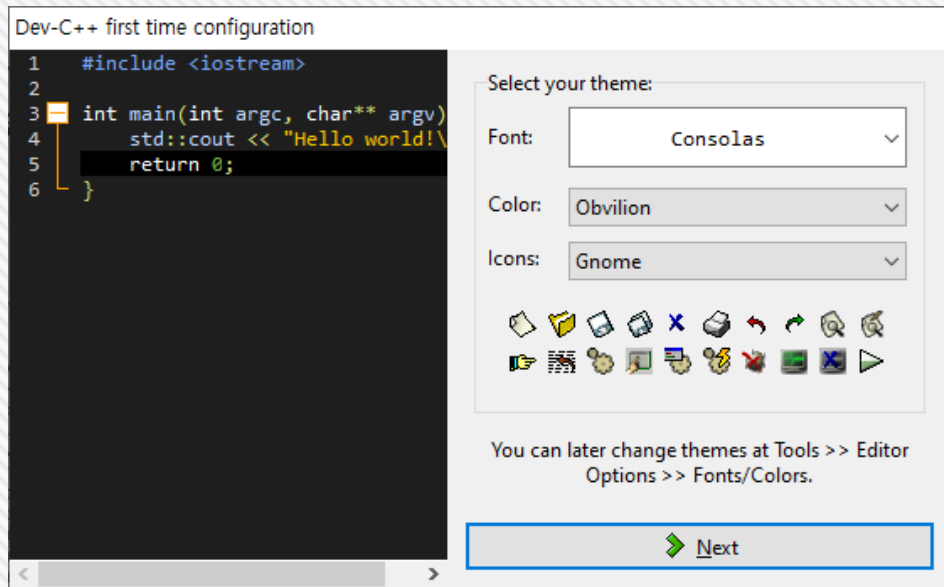
5. 첫번째 프로그램

1. 프로그램 기초 1-1. 프로그램 이란

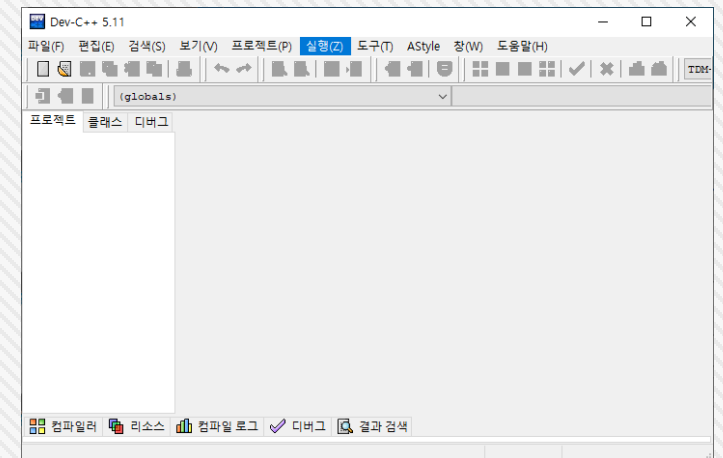
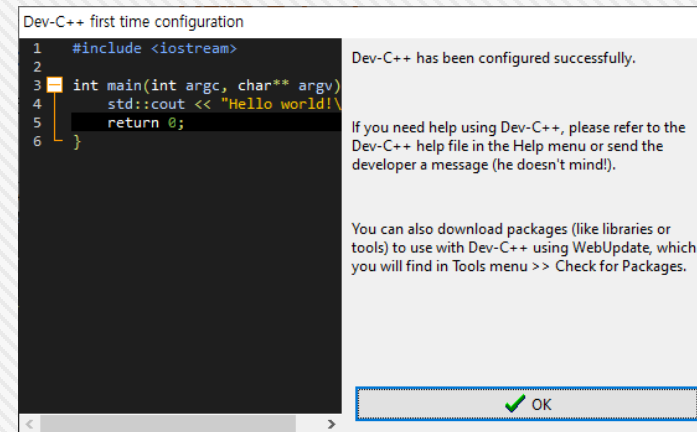
“ Hello C 언어”

Dev-C ++ 실행 후 최초 환경 설정

3. 원하는 테마 선택 후 Next 클릭



4. 설정 완료 ok 클릭 하면 개발 화면 오픈됨



5. 첫번째 프로그램

1. 프로그램 기초 1-1. 프로그램 이란

C 언어 표현 방법

- 구문이 끝날 때 ; (세미콜론) -> `printf("Hello, world\n");`
- 주석: 컴파일러가 처리하지 않으므로 프로그램의 실행에는 영향을 주지 않음
 - 한 줄 주석: `//`
 - 범위 주석: `/* */`
- 여러 문법에서 {} (중괄호)를 많이 사용하는데 보통 중괄호는 코드의 범위

```
#include <stdio.h>

/*
 * 범위 주석 .....
 * 프로그램 : C 언어 표현
 * 작 성 자 : 홍길동
 * 작성일자 : 2021.12.30
 */

int main() {
    char a;           // 블록 시작
    int num = 1;       // 변수 선언
    a = 'a';           // 변수 선언과 동시에 할당
                      // 데이터 할당

    // 한줄 주석
    printf("Hello, world! %c \n" , a); // 한 줄 주석 출력 결과 : Hello, world! a

    /*
     * 범위 주석에
     * \n은 개행 문자 입니다. */
    if (a > 10) { // if 블록 시작
        printf("a \n");
    } // if 블록 끝

    for ( num = 0; num < 10; num++) {
        printf("Hello, world!\n"); // Hello, world! 10번 출력됨
    }
    return 0;
} // 블록 끝
```

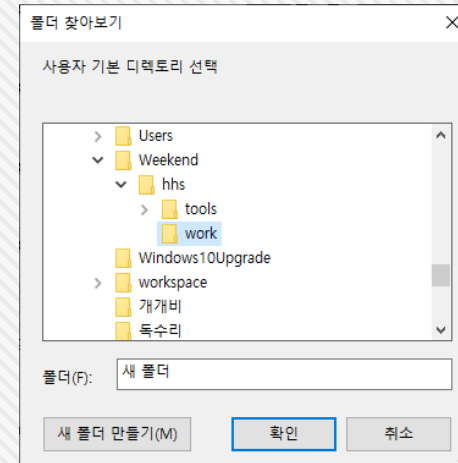
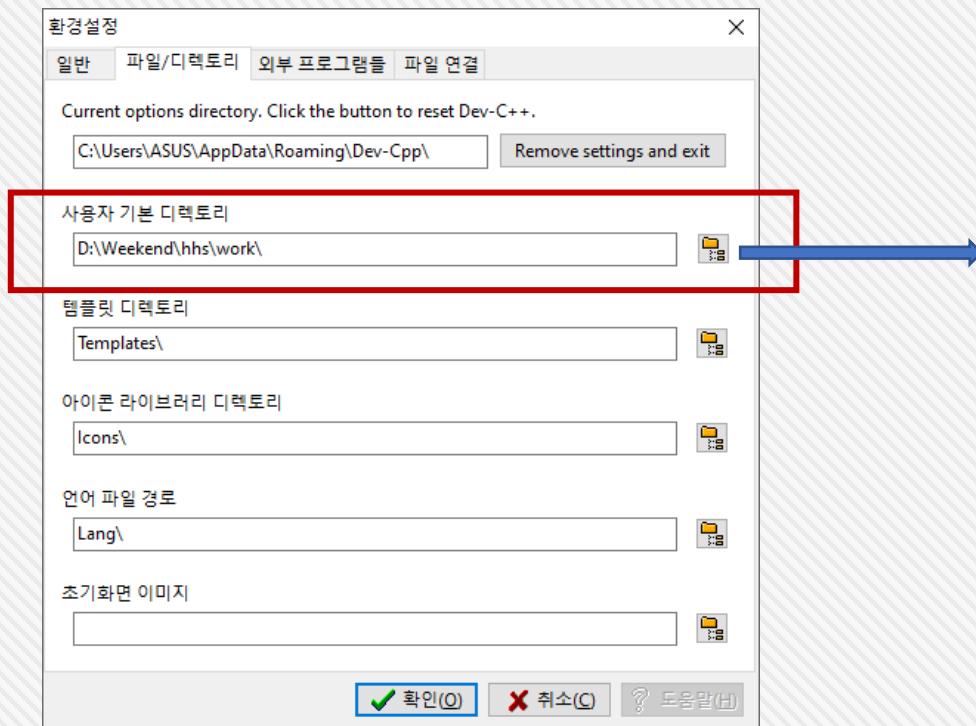
5. 첫번째 프로그램

1. 프로그램 기초 1-1. 프로그램 이란

“ Hello C 언어”

Dev-C ++ 실행 후 최초 환경 설정

5. 작업 폴더 변경
메뉴 : 도구 > 환경설정



D:\Weekend\ 영문이름 \work

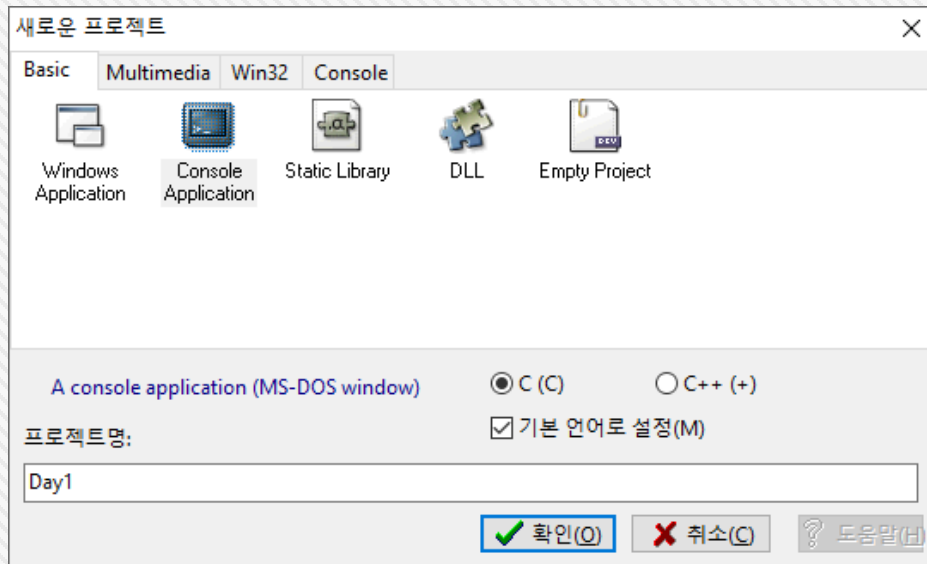
5. 첫번째 프로그램

1. 프로그램 기초 1-1. 프로그램 이란

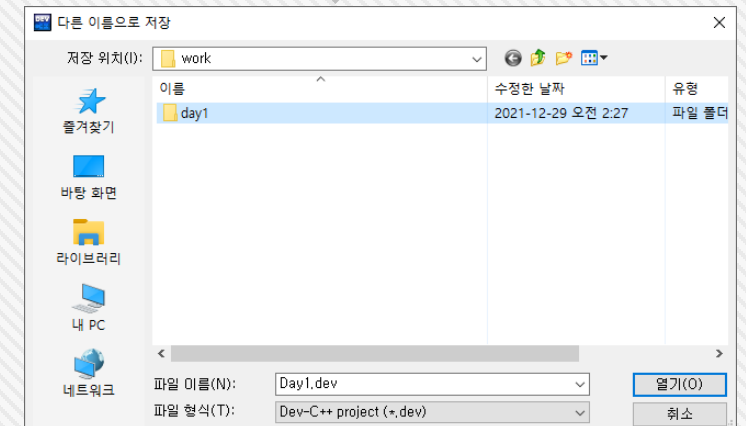
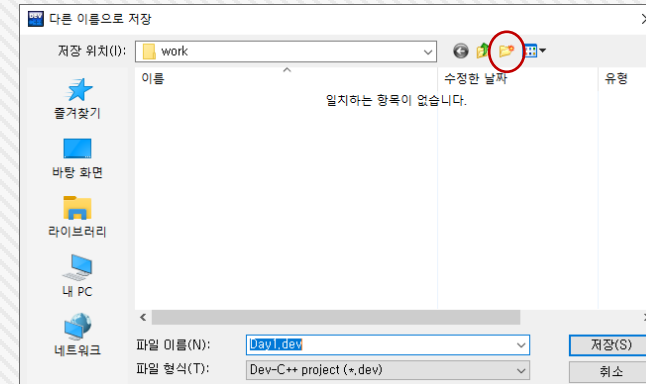
“ Hello C 언어 ”

첫번째 프로그램 작성

1. 프로젝트 생성 : 메뉴 :: 파일 > 프로젝트



2. 프로젝트 폴더 생성
: 폴더 생성 : day1 후 열기 버튼 클릭



5. 첫번째 프로그램

1. 프로그램 기초
1-1. 프로그램 이란

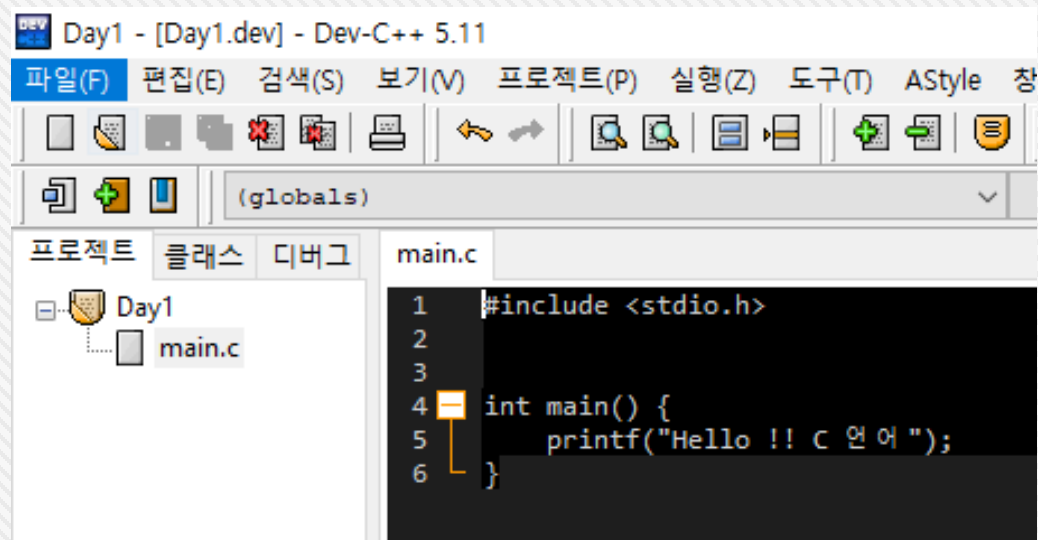
“ Hello C 언어”

첫번째 프로그램 작성

1. 작업 창에 다음과 같이 작성

```
#include <stdio.h>

int main() {
    printf("Hello !! C 언어");
}
```



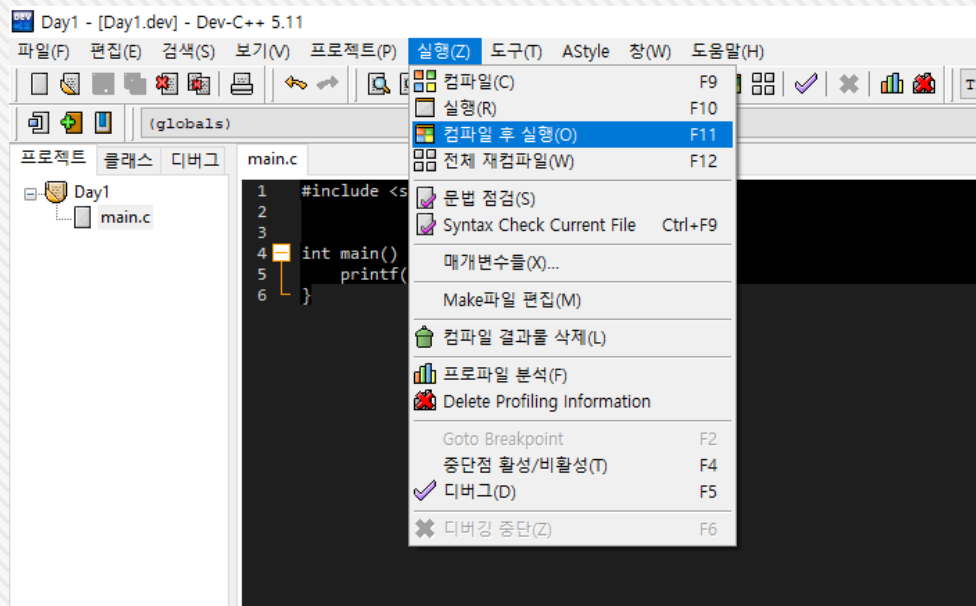
5. 첫번째 프로그램

1. 프로그램 기초 1-1. 프로그램 이란

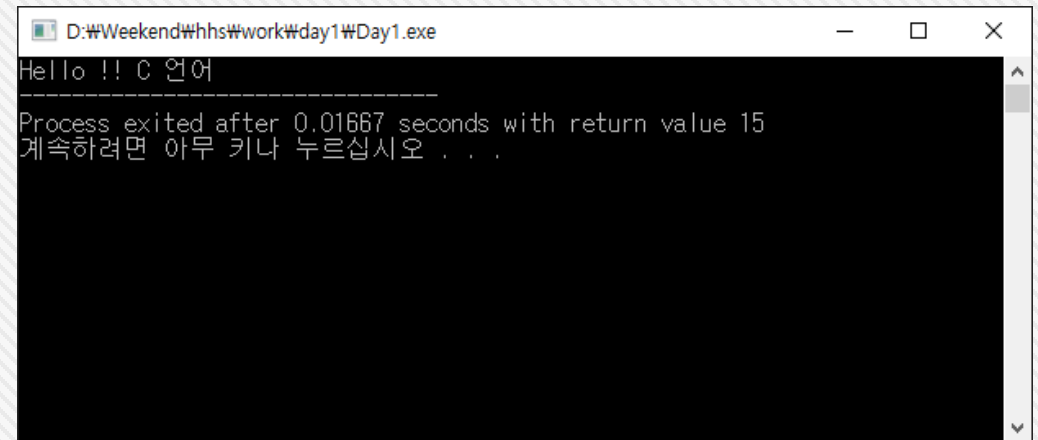
“ Hello C 언어”

첫번째 프로그램 작성

2. 실행 : 메뉴 : 실행 > 컴파일 후 실행



2. 실행 결과



1. 프로그램 기초

1-1. 프로그램이란

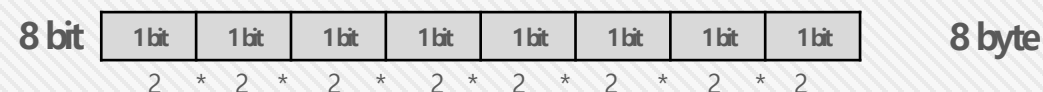
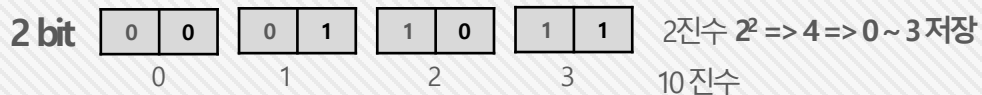
데이터 저장 장소

- 각 주소가 담긴 일종의 긴 띠



Bit, Byte

- bit: 0, 1을 저장 할 수 있는 공간



- 1 bit 증가 할 때 마다 저장 단위는 2의 배수

음수와 양수 구분

- 부호 bit를 사용 하여 음수, 양수를 구분 함

MSD(Most Significant Digit)

LSD(Least Significant Digit)



0:양수 2 * 2 * 2 * 2 * 2 * 2 * 2 = 2⁷ = 128
1:음수

양수: $2^7 = 128 = 0 \sim 127$, 음수: $-128 \sim -1 \Rightarrow -128 \sim 128$ 사이 중 하나 저장

- 양수 최대값 : 01111111 = 127 양수 최소값 : 00000000 = 0

- 음수 최대값 : 11111111 = -1 음수 최소값 : 10000000 = -128

자리값	$2^7=64$	$2^6=32$	$2^5=16$	$2^4=8$	$2^3=4$	$2^2=2$	1
이진수	1	0	1	1	1	1	1
10진수	64	0	16	8	4	2	1

양수: 95
음수: -33

컴퓨터 내부에서는 사칙연산을 할 때 덧셈을 담당하는 가산기(Adder)만 이용하기 때문에 뺄셈은 덧셈으로 형식을 변환하여 계산

즉 $A-B=A+(B\text{의 보수})$

- 보수: 두 수의 합이 진법의 밑수(N)가 되게 하는 수
- 1의 보수: 1010 -> 0101
- 2의 보수: 1의 보수에 1을 더한 값: 1010 -> 0101(1의 보수) + 0001 = 0110 = 6
- 10진수를 2진수 변환:

	7
뭇을 2로 나함	뭇 3 나머지 1
뭇을 2로 나함	뭇 1 나머지 1
뭇을 2로 나함	뭇 0 나머지 1

6. 자료 저장 방식

1. 프로그램 기초 1-1. 프로그램 이란

“ ASCII 코드는 문자를 숫자로 표현 하기 위한 약속 ”

아스키코드

- ASCII (American Standard Code for Information Interchange, 미국 정보 교환 표준 부호)
- 1963년 미국 ANSI에서 표준화한 정보교환용 7비트 부호체계
- 000(0x00)부터 127(0x7F)까지 총 128개의 부호가 사용
- 나머지 1비트를 통신 에러 검출을 위한 용도 (Parity Bit : CRC 체크섬)
-> 전송 도중 신호가 변질된 것을 수신측에서 검출
- 한글 : UTF-8을 사용해야 깨짐을 막을 수 있음 (2 바이트)
- 현재는 확장 하여 255개를 사용함
- 문자의 ASCII 값은 부호 없는 1바이트에 저장

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]	48	30	110000	60	0	96	60	1100000	140	`
1	1	1	1	[START OF HEADING]	49	31	110001	61	1	97	61	1100001	141	a
2	2	10	2	[START OF TEXT]	50	32	110010	62	2	98	62	1100010	142	b
3	3	11	3	[END OF TEXT]	51	33	110011	63	3	99	63	1100011	143	c
4	4	100	4	[END OF TRANSMISSION]	52	34	110100	64	4	100	64	1100100	144	d
5	5	101	5	[ENQUIRY]	53	35	110101	65	5	101	65	1100101	145	e
6	6	110	6	[ACKNOWLEDGE]	54	36	110110	66	6	102	66	1100110	146	f
7	7	111	7	[BELL]	55	37	110111	67	7	103	67	1100111	147	g
8	8	1000	10	[BACKSPACE]	56	38	111000	70	8	104	68	1101000	150	h
9	9	1001	11	[HORIZONTAL TAB]	57	39	111001	71	9	105	69	1101001	151	i
10	A	1010	12	[LINE FEED]	58	3A	111010	72	:	106	6A	1101010	152	j
11	B	1011	13	[VERTICAL TAB]	59	3B	111011	73	;	107	6B	1101011	153	k
12	C	1100	14	[FORM FEED]	60	3C	111100	74	<	108	6C	1101100	154	l
13	D	1101	15	[CARRIAGE RETURN]	61	3D	111101	75	=	109	6D	1101101	155	m
14	E	1110	16	[SHIFT OUT]	62	3E	111110	76	>	110	6E	1101110	156	n
15	F	1111	17	[SHIFT IN]	63	3F	111111	77	?	111	6F	1101111	157	o
16	10	10000	20	[DATA LINK ESCAPE]	64	40	1000000	100	@	112	70	1110000	160	p
17	11	10001	21	[DEVICE CONTROL 1]	65	41	1000001	101	A	113	71	1110001	161	q
18	12	10010	22	[DEVICE CONTROL 2]	66	42	1000010	102	B	114	72	1110010	162	r
19	13	10011	23	[DEVICE CONTROL 3]	67	43	1000011	103	C	115	73	1110011	163	s
20	14	10100	24	[DEVICE CONTROL 4]	68	44	1000100	104	D	116	74	1110100	164	t
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]	69	45	1000101	105	E	117	75	1110101	165	u
22	16	10110	26	[SYNCHRONOUS IDLE]	70	46	1000110	106	F	118	76	1110110	166	v
23	17	10111	27	[ENG OF TRANS. BLOCK]	71	47	1000111	107	G	119	77	1110111	167	w
24	18	11000	30	[CANCEL]	72	48	1001000	110	H	120	78	1111000	170	x
25	19	11001	31	[END OF MEDIUM]	73	49	1001001	111	I	121	79	1111001	171	y
26	1A	11010	32	[SUBSTITUTE]	74	4A	1001010	112	J	122	7A	1111010	172	z
27	1B	11011	33	[ESCAPE]	75	4B	1001011	113	K	123	7B	1111011	173	{
28	1C	11100	34	[FILE SEPARATOR]	76	4C	1001100	114	L	124	7C	1111100	174	}
29	1D	11101	35	[GROUP SEPARATOR]	77	4D	1001101	115	M	125	7D	1111101	175	~
30	1E	11110	36	[RECORD SEPARATOR]	78	4E	1001110	116	N	126	7E	1111110	176	~
31	1F	11111	37	[UNIT SEPARATOR]	79	4F	1001111	117	O	127	7F	1111111	177	[DEL]
32	20	1000000	40	[SPACE]	80	50	1010000	120	P					
33	21	1000001	41	!	81	51	1010001	121	Q					
34	22	1000100	42	"	82	52	1010010	122	R					
35	23	1000101	43	#	83	53	1010011	123	S					
36	24	1001000	44	\$	84	54	1010100	124	T					
37	25	1001001	45	%	85	55	1010101	125	U					
38	26	1001010	46	&	86	56	1010110	126	V					
39	27	1001011	47	'	87	57	1010111	127	W					
40	28	1010000	50	(88	58	1011000	130	X					
41	29	1010001	51)	89	59	1011001	131	Y					
42	2A	1010100	52	*	90	5A	1011010	132	Z					
43	2B	1010101	53	+	91	5B	1011011	133	[
44	2C	1011000	54	,	92	5C	1011100	134	\					
45	2D	1011001	55	-	93	5D	1011101	135]					
46	2E	1011100	56	.	94	5E	1011110	136	^					
47	2F	1011101	57	/	95	5F	1011111	137	_					

7. 자료형

1. 프로그램 기초 1-1. 프로그램 이란

“ 데이터는 크기에 맞게 메모리에 저장 되는데 이때 몇 바이트를 사용 할 것을 명시 하는 것 ”

데이터 타입 (DataType) or 자료형

- 데이터를 메모리에 저장 하기 위한 단위로 데이터의 종류에 따라서 할당 하는 메모리의 크기를 명시적 해야 하는 것
- C 언어는 예약어로 제공(Built-in DataType)과 사용자가 정의(User-Defined DataType)가 있다.
- signed 부호 있는 데이터로 생략 가능

자료형			설명	바이트	범위
정수형	부호있음	short	short형 정수	2	-32768 ~ 32767
		int	정수	4	-2147483648 ~ 2147483647
		long	long형 정수	4	-2147483648 ~ 2147483647
	부호없음	unsigned short	부호없는 short형 정수	2	0 ~ 65535
		unsigned int	부호없는 정수	4	0 ~ 4294967295
		unsigned long	부호없는 long형 정수	4	0 ~ 4294967295
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

- signed 정수형 : https://github.com/hyomee/c_basic/blob/main/src/datatype/45_01.c
- unsigned 정수형 : https://github.com/hyomee/c_basic/blob/main/src/datatype/45_02.c
- signed 정수형 최대값/최소값 : https://github.com/hyomee/c_basic/blob/main/src/datatype/45_03.c
- unsigned 정수형 최대값, 최소값 : https://github.com/hyomee/c_basic/blob/main/src/datatype/45_04.c
- 크기 구하기 : https://github.com/hyomee/c_basic/blob/main/src/datatype/sizeof.c

“ 데이터는 크기에 맞게 메모리에 저장 되는데 이때 몇 바이트를 사용 할 것을 명시 하는 것 ”

정수형

- **Type**: char, short, int 사용, **부호**: 부호 있음: signed (생략가능: MSD [부호bit]), 부호없음 unsigned
- 부호, type 를 조합 하여 사용

```
int main()
{
    char num1 = -10;                // 1바이트 부호 있는 정수형으로 변수를 선언하고 값 할당
    short num2 = 30000;             // 2바이트 부호 있는 정수형으로 변수를 선언하고 값 할당
    int num3 = -1234567890;         // 4바이트 부호 있는 정수형으로 변수를 선언하고 값 할당
    long num4 = 1234567890;         // 4바이트 부호 있는 정수형으로 변수를 선언하고 값 할당
    long long num5 = -1234567890123456789; // 8바이트 부호 있는 정수형으로 변수를 선언하고 값 할당

    // char, short, int는 %d로 출력하고 long은 %ld, long long은 %lld로 출력
    printf("%d %d %d %d %lld\n", num1, num2, num3, num4, num5);
    // -10 30000 -1234567890 1234567890 -1234567890123456789

    unsigned char unNum1 = 200;     // 1바이트 부호 없는 정수형으로 변수를 선언하고 값 할당
    unsigned short unNum2 = 60000;   // 2바이트 부호 없는 정수형으로 변수를 선언하고 값 할당
    unsigned int unNum3 = 4123456789; // 4바이트 부호 없는 정수형으로 변수를 선언하고 값 할당
    unsigned long unNum4 = 4123456789; // 4바이트 부호 없는 정수형으로 변수를 선언하고 값 할당
    unsigned long long unNum5 = 12345678901234567890; // 8바이트 부호 없는 정수형으로 변수를 선언하고 값 할당

    // unsigned char, unsigned short, unsigned int는 %u로 출력하고
    // unsigned long은 %lu, unsigned long long은 %llu로 출력
    printf("%u %u %u %u %llu\n", unNum1, unNum2, unNum3, unNum4, unNum5);
    // 200 60000 4123456789 4123456789 12345678901234567890
}
```

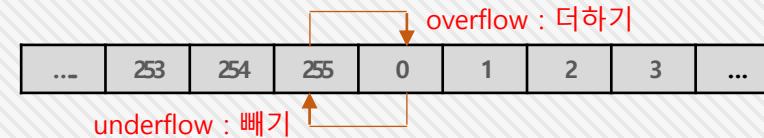
“ 범위에 벗어나면 범위 순회 ”

정수형 범위를 벗어난 경우

- 선언한 type의 범위를 벗어난 값을 할당 하면 범위를 순회 함.
- signed char: 1byte (부호 (MSD) 1bit, 데이터: 7 bit) : -128~127



- unsigned char: 1byte (데이터: 8 bit) : 0 ~ 255



```
int main()
{
    char singendCharOverFlow = 128;
    char singendCharUnderFlow = -129;

    unsigned char unsingendCharOverFlow = 256;
    unsigned char unsingendCharUnderFlow = -1;

    printf("%d %d %u %u \n", singendCharOverFlow, singendCharUnderFlow, unsingendCharOverFlow, unsingendCharUnderFlow);
    // -128 127 0 255

    return 0;
}
```

- 벗어난 범위: https://github.com/hyomee/c_basic/blob/main/src/datatype/47_01.c

“ 정수형 char을 사용 하여 1문자 저장 ”

문자형

- 정수 자료형 인 char를 이용하여 문자 한 개를 저장
- 문자 자체를 저장하는 것이 아니라 문자에 해당하는 정수값을 저장 -> 아스키(ASCII) 코드
- 65(0x41)부터 90(0x5A)까지 A~Z 알파벳 대문자를 나타내고, 97(0x61)부터 122(0x7A)까지 a~z 알파벳 소문자
- 문자는 ' ' (작은따옴표)로 묶어서 표현 : char a = 'a'; char a = 'hello world' (잘못된 표현) -> char a[] = 'hello world' ;

```
int main()
{
    char c1 = 'a';                // 문자 변수를 선언하고 문자 a를 저장
    char c2 = 'b';                // 문자 변수를 선언하고 문자 b를 저장
    char c3[] = "hello 안녕";    // 문자 배열 변수 선언 하고 문자 hello 안녕 저장 "

    // char를 %c로 출력하면 문자가 출력되고, %d로 출력하면 정수값이 출력됨
    printf("%c, %d\n", c1, c1);    // a, 97: a의 ASCII 코드값은 97
    printf("%c, %d\n", c2, c2);    // b, 98: b의 ASCII 코드값은 98
    printf("%s, %d\n", c3, c3);    // hello 안녕, ASCII 코드값은 6487568

    printf("%c %d\n", 'a' + 1, 'a' + 1); // b 98: a는 ASCII 코드값 97이고, 97에 1을 더하여 98이 되었으므로 b가 출력됨
    printf("%c %d\n", 97 + 1, 97 + 1); // b 98: ASCII 코드값 97에 1을 더하여 98이 되었으므로 b가 출력됨
    return 0;
}
```

- 문자형: https://github.com/hyomee/c_basic/blob/main/src/datatype/4&c

“ 실수형은 지수표기법으로 저장 ”

실수형 - 소수점

- **Type** : float (4 byte : 32bit), double (8byte : 64bit) 사용, IEEE 754 규격에 규약에 정의된 부동 소수점 표현
- 정수형에 비해서 속도가 떨어 진다.
- **overflow** : INF (무한대(infinity)) , **underflow** : 0 으로 됨

*지수 표기법 (과학적 표기법)

- 실수e지수 : 실수 * 10의 거듭제곱. -> 21e+3이라면 21 * 1000 = 2100
- 실수e지수 : 실수 * (1 / 10의 거듭제곱) -> 21e-2라면 21 * (1/100) = 0.21

• 32bit 부동 소수점 : float

부호 (1 bit)	지수부 (8 bit)	가수부 (23 bit)
- 양수: 0 - 음수: 1	- -126 ~ 128 - 00000000: -127 -> 오류나 특수 상황에 사용 - 편향차수: 127 (0) -> 실수는 부호비트를 사용하지 않고 이 값을 기준으로 함	

* 실수 : 567.984 : $5.67984 * 10^2$ (지수 표기법으로 저장) : 5.67984E2
- 5.67984 : 유효 숫자 가수부, 10^2 : 자리수, 2 : 지수부

• 64bit 부동 소수점 : float

부호 (1 bit)	지수부 (11 bit)	가수부 (52 bit)
- 양수: 0 - 음수: 1	- -1022 ~ 1024 - 편향차수: 1023 (0) -> 실수는 부호비트를 사용하지 않고 이 값을 기준으로 함	

```
int main()
{
    float num1 = 0.1f;           // 32bit 부동소수점 변수를 선언하고 값을 할당 * float는 숫자 뒤에 f를 붙임
    double num2 = 3867.215820;   // 64bit 부동소수점 변수를 선언하고 값을 할당 * double은 숫자 뒤에 아무것도 붙이지 않음
    long double num3 = 9.327513l; // 64bit 부동소수점 변수를 선언하고 값을 할당 * long double은 숫자 뒤에 l을 붙임

    // float와 double은 %f로 출력, long double은 %Lf로 출력
    printf("%f %f %Lf\n", num1, num2, num3);
    // 0.100000 3867.215820 0.000000

    return 0;
}
```

- 실수형 : https://github.com/hyomee/c_basic/blob/main/src/datatype/48.c
- 실수형 범위 : https://github.com/hyomee/c_basic/blob/main/src/datatype/49.01.c

Content

II. 상수, 변수, 함수

1. 상수
2. 변수
3. 함수

1. 숫자 상수

2. 상수, 변수, 함수

2-1. 상수

“ 프로그램 실행 중 값이 한번 결정 되면 변하지 않는 정보 ”

상수

- 변하지 않는 값
- 리터럴 (literal) : 수나 문자와 같은 값 자체
- **const** 를 사용 해서 상수를 만든다

상수 종류

- 숫자 상수
- 문자 상수
- 논리 상수

`const int 상수 age = 리터럴 30;`

숫자 상수 (Numeric Constant)

- 정수형 상수 : -5, 0, 10
- 실수형 상수 : 0.1, -5.1, 235.334, 지수표기법 사용 : 0.15e+3, 1.34e-5
- 사용법

```
int price = 1000;           // 정수형상수 1000을 부호가 있는 int 자료형 price변수에 저장
unsigned int count = 0;     // 정수형상수 0을 부호가 없는 int 자료형 count에 저장

float a = 0.17;             // 실수형상수 0.17을 float 자료형 a 변수에 저장
```

- 8진수 : 숫자 앞에 0(숫자)을 붙임

```
int b = 0275;
// 64(82)*2+7*8(81) + 5(80) = 189

printf("%d %#o \n", b, b); // 189 0275
```

8진수는 범위밖에 있는 숫자 표현 시 오류 발생 -> 0912 [Error] !
nvalid digit "9" in octal constant -> 8진수의 범위는 0 ~ 7

- 16진수 : 숫자 앞에 0x붙임

```
int b = 0x275;
// 256(162)*2+7*16(161) + 5(160) = 629

printf("%d %#x \n", c, c); // 629 0x275
```

- 리터럴 : <https://github.com/hyomee/c-basic/blob/main/src/variable/literal.c>
- 상수 : <https://github.com/hyomee/c-basic/blob/main/src/variable/const.c>

2. 문자/문자열 상수

2. 상수, 변수, 함수

2-1. 상수

“ 단일 함수 작은 따옴표, 문자열 함수 큰 따옴표 ”

문자형 상수 (Character Constant)

- 작은 따옴표 (' ') 로 표시 - 단일 문자, 영문자('a' ~ 'z' , 숫자('1' , '123'), 특수문자('*', '# ' ..) 사용
- 사용법

```
char ch1 = 'A';           // A의 ASCII 값 65를 변수 저장
char ch2 = 'A' + 1;       // A의 ASCII 값 65 + 1 = 66을 변수 저장
printf("%c %d\n", ch1, ch1); // A 65
printf("%c %d\n", ch2, ch2); // B 66
```

문자열형 상수 (Character-string Constant)

- 큰 따옴표 (" ") 로 표시
- 문자열이 단일 문자라도 NULL이 마지막에 있으므로 단일문자와 다르다.
- 영문자는 1 Byte, 한글 2byte

Hello는 6바이트

'H'	'e'	'l'	'l'	'o'	'\0'
-----	-----	-----	-----	-----	------

실제는 마지막 null(0000) 포함 7byte

```
char ch1[] = "대한민국"; // 한글 2byte로 null문자 포함 크기는 9
char ch2[] = "Hello";    // 영문자 1byte로 null문자 포함 크기는 6
char ch3[] = "1" ;       // 숫자를 문자로 취급 하여 1byte 로 null문자 포함 크기는 2
char ch4  = '1' ;        // 단일문자 1byte 로 크기는 1
char ch5  = "1" ;
printf("%s , %d , %d\n", ch1, ch1, sizeof(ch1)); // 대한민국 , 6487568 , 9
printf("%s , %d , %d\n", ch2, ch2, sizeof(ch2)); // Hello , 6487552 , 6
printf("%s , %d , %d\n", ch3, ch3, sizeof(ch3)); // 1 , 6487536 , 2
printf("%c , %d , %d\n", ch4, ch4, sizeof(ch4)); // 1 , 49 , 1
//printf("%c , %d , %d\n", ch5, ch5, sizeof(ch5)); // initialization makes integer from pointer without a cast
printf("%s , %d , %d\n", ch5, ch5, sizeof(ch5)); // (null) , 0 , 1
```

“ 참/거짓 값을 가지는 것 ”

▲ 논리 상수 (Boolean Constant)

- 참 : TRUE, 0(숫자)이 아닌 값, 거짓 : FALSE, 0(숫자)
- 사용법

```
#include <stdio.h>

#define TRUE 1
#define FALSE 0

int main() {
    if (TRUE) {
        printf("논리 함수 TRUE, 0이아닌 값");
    } else {
        printf("논리 함수 FALSE, 0");
    }

    return 0;
}
```

1. 변수

2. 상수, 변수, 함수 2-2. 변수

“ 변수 선언은 메모리 공간을 할당 받는 것이고 초기화는 초기값을 넣는 것 ”

변수

- 프로그램 수행 중 변하는 값
- 프로그램 수행 중 데이터를 처리 하기 위한 저장 공간

변수명 규칙

- 영문 문자와 숫자 조합
- 대소문자 구분
- 문자로 시작, 숫자로 시작 할 수는 없음
- 특수 문자는 사용 하지 않으며, _(밑줄문자)는 사용 가능
- C언어의 키워드(int, if, for...)는 사용 할 수 없음
- 공백 문자는 사용 할 수 없음
- 의미 있는 이름으로 작명 해야 함

올바른 사용 : name,, _price, DsCount, dsSubscriber, my_car, myCar
잘못된 사용 : 1000, lab, aaaa, 7asss, 7_dsCount, ds Sub,

표기법:

- 카멜 표기법: 소문자로 시작 하고 연결 단어를 대문자로 표시
예) myName, myHome
- 헝가리안 표기법: 처음 문자에 의미(자료형)를 부여 - 접두사 사용
예) i_count: int 형 변수, f_price: 실수형 변수

변수 선언

- 자료형 + 변수명 + ;로 선언
- 자료형에 따라서 크기가 결정 된다.
- 변수는 컴파일 이후 메모리 주소를 의미 즉 할당된 메모리 공간의 이름

자료형

int

변수명

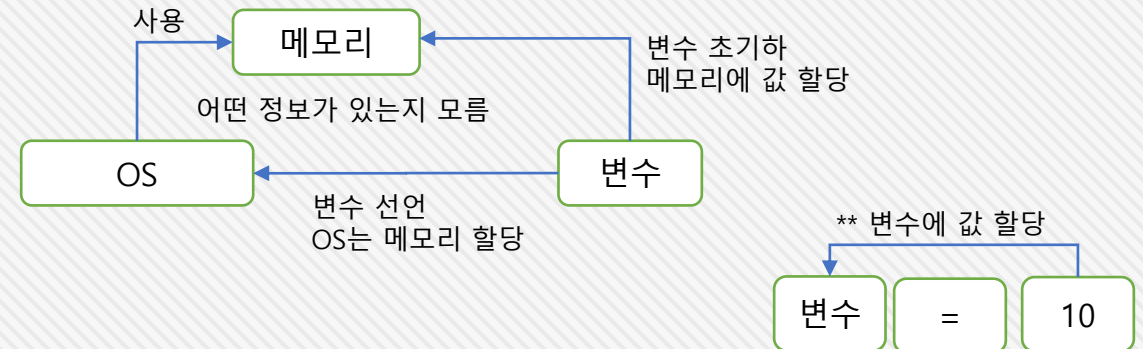
num ;

-> 변수가 임의의 메모리 공간을 자료형 만큼 할당 받는다



할당, 변수 초기화

- 할당: 어떤 메모리 공간을 임의로 사용할 수 있도록 주는 것
- 메모리는 운영체계가 부여 하는 것으로 어떤 정보가 있는지 알 수 없음으로 메모리 할당 받은 후 값을 설정 하는 것을 변수 초기화라 함



2. 변수 예제

2. 상수, 변수, 함수

2-2. 변수

변수를 만들고 저장 후 출력 한다

예제1) 변수 선언과 초기화 분리

```
#include <stdio.h>

// 선언과 초기화 분리
int main() {
    int age;           // 변수선언
    age = 30;          // 변수 초기화
    printf("%d", age); // 30
}
```

예제3) 변수 여러 개를 한번에 선언

```
#include <stdio.h>

int main() {
    int age, price, count; // 변수 여러개를 한번에 선언
    age = 30;
    price = 1000;
    count = 1;
    printf("%d, %d, %d", age, price, count);
}
```

예제12) 변수 선언과 동시에 초기화

```
#include <stdio.h>

// 선언과 동시에 초기화
int main() {
    int age = 30;           // 변수선언후 초기화
    printf("%d", age);      // 30
}
```

예제4) 변수 여러 개를 한번에 선언, 초기화 동시

```
#include <stdio.h>

int main() {
    int age=30, price, count;
    price = 1000;
    count = 1;

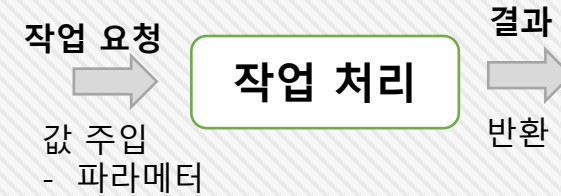
    printf("%d, %d, %d", age, price, count);
}
```

- 예제1 : https://github.com/hyomee/c_basic/blob/main/src/variable/variable01.c
- 예제2 : https://github.com/hyomee/c_basic/blob/main/src/variable/variable02.c
- 예제3 : https://github.com/hyomee/c_basic/blob/main/src/variable/variable03.c
- 예제4 : https://github.com/hyomee/c_basic/blob/main/src/variable/variable04.c

“ 하나의 기능을 처리 하기 위한 명령문의 그룹 ”

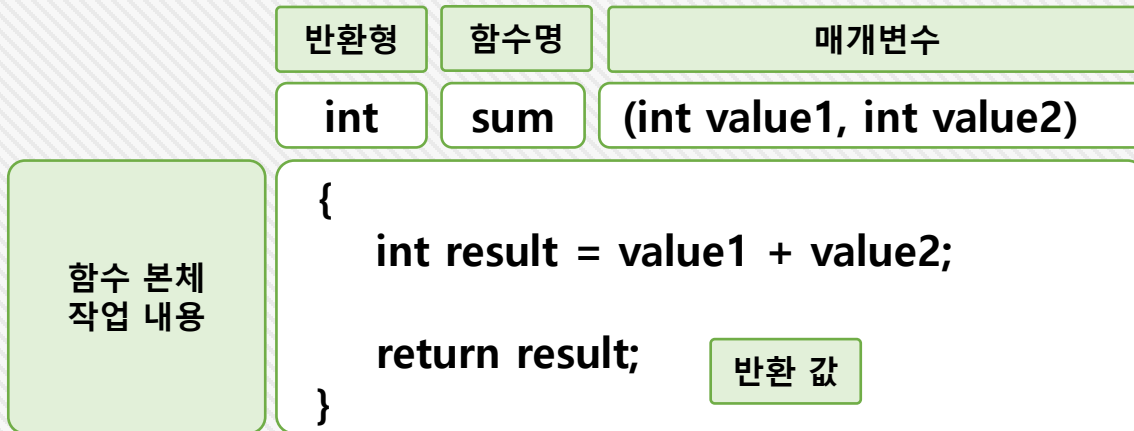
함수

- 하나의 기능을 처리 하기 위해서 실행 순서대로 명령어 작성
- 함수를 만들지 않으면
 - 소스가 지저분 해지고, - 작성 후 수정이 어렵고, - 비슷한 처리를 하는 많아 진다.



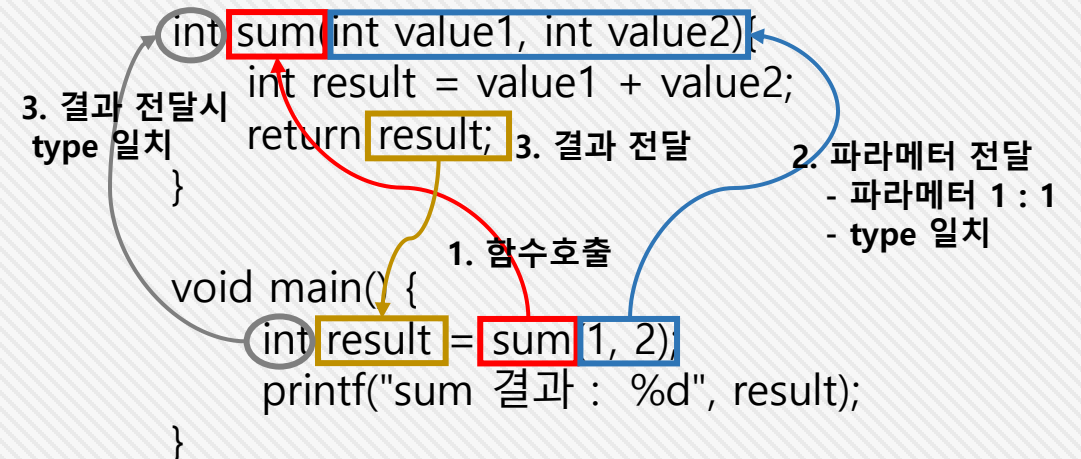
함수 구조

- 반환형, 함수명, 매개변수, 본체, 반환 값으로 구성 된다.



함수 호출 과정

- 반환 값이 없는 경우는 void로 반환 형 선언



2. 함수 작성시 주의 사항

함수 작성시 주의 사항

- 반환 형을 void로 선언

```
void sum(int value1, int value2){
    int result = value1 + value2;
    printf("sum 결과 : %d", result);
}
void main() {
    sum(1, 2);
}
```

- return 문장 이후에 명령어가 있으면 실행 되지 않는다.

```
int sum(int value1, int value2){
    int result = value1 + value2;
    return result;
    result = 10;           // 실행되지 않음
    return result;         // 실행되지 않음
}
void main() {
    int result = sum(1, 2);
    printf("sum 결과 : %d", result);
}
```

- 함수명은 변수명 규칙과 동일 하다.

- 호출자가 소스상에 아래에 있어야 함

- 함수가 정의되지 않았다고 하면서 컴파일 경고와 에러가 발생
=> 함수 원형을 선언 하여 해결함
=> 함수의 파라미터는 생략 할 수 있음.

컴파일러의
소스 번역
방향

```
#include <stdio.h>

int sum(value1, value2); /* 함수 원형 */

void main() {
    int result = sum(1, 2);
    printf("sum 결과 : %d", result);
}

int sum(int value1, int value2){
    int result = value1 + value2;
    return result;
}
```

- 반환 형을 void로 선언 : https://github.com/hyomee/c_basic/blob/main/src/function/void.c
 - return 문장 : https://github.com/hyomee/c_basic/blob/main/src/function/twoReturn.c
 - 함수원형 : https://github.com/hyomee/c_basic/blob/main/src/function/functionPrototype.c

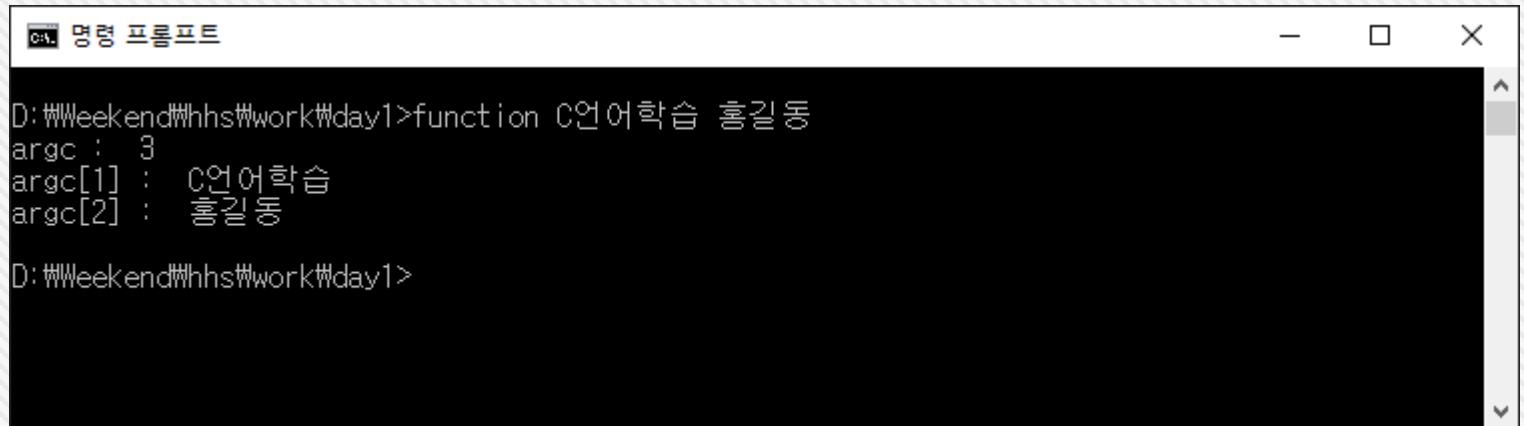
3. 실행 파일 수행시 파라미터 받기

2. 상수, 변수, 함수
2-3. 함수

실행 파일 수행 시 파라미터 받기

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

```
void main(char argc, char *argv[]) {  
    printf("argc : %d\n", argc);           // argc 실행인자의 갯수  
    printf("argv[1] : %s\n", argv[1]);     // argv 파라미터  
    printf("argv[2] : %s\n", argv[2]);  
}
```



```
cmd 명령 프롬프트  
D:\Weekend\hhs\work\day1>function C언어학습 홍길동  
argc : 3  
argv[1] : C언어학습  
argv[2] : 홍길동  
D:\Weekend\hhs\work\day1>
```

- 파라미터 : https://github.com/hyomee/c_basic/blob/main/src/function/runArg.c

Content

III. 라이브러리

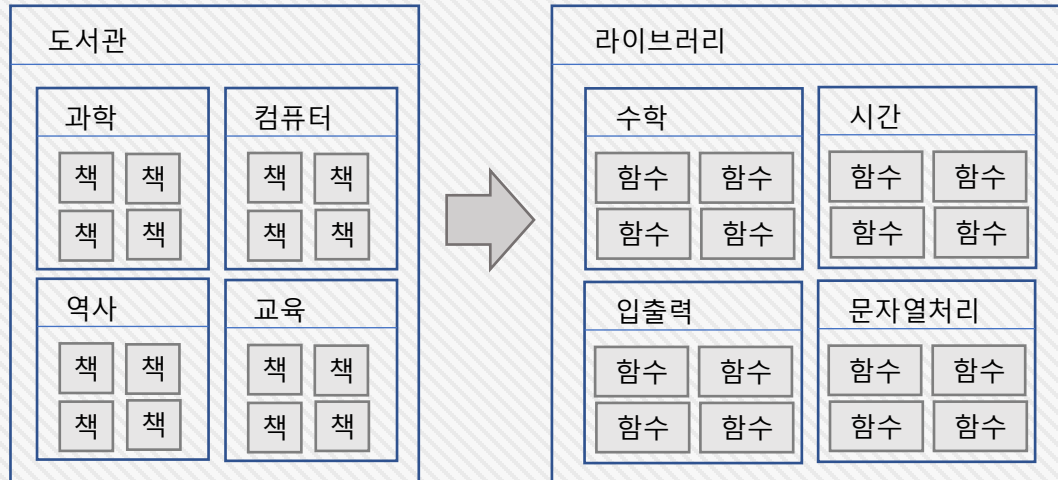
1. 라이브러리
2. 전처리기 -> 고급 매크로
3. 표준 라이브러리
4. 표준 입출력 함수
5. CodeUp 문제 풀이

1. 라이브러리

3. 라이브러리 3-1. 라이브러리

라이브러리

- 비슷한 기능을 모아 놓은 것



라이브러리 종류

- 표준 라이브러리

: 핵심 컴파일러 패키지의 일부로 포함 된 일련의 함수, 상수 및 기타 언어 정의

- 런타임 라이브러리

: 런타임 환경의 몇몇 행동들을 유발하기 위해서 컴파일러에 의해

사용되는 저수준 루틴들의 집합 (운영 체제 지원을 받는 함수)

: 컴파일러와 플랫폼에 특화되어 있음

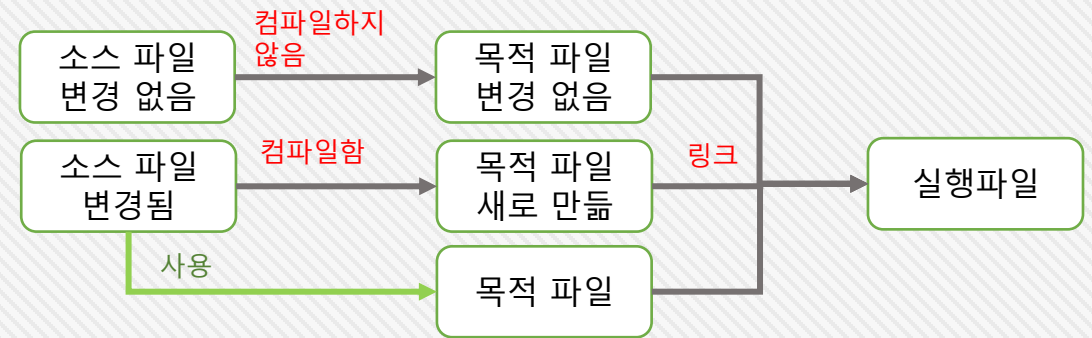
: windows (DLL), linux (so)

: printf 와 같은 표준 C 라이브러리

라이브러리 사용 이유

- 모든 기능을 직접 코딩하지는 못하기 때문에
- 시간을 절약하기 위해

라이브러리 등장 한 이유



- 변경이 없는 소스 파일을 "소스 전체 컴파일" 을 할 때 변경이 없어도 컴파일을 하므로 효율적 이지 못함 이런 단점을 피하기 위해서 미리 컴파일한 목적 파일 (라이브러리) 을 링크만 할 때 사용 함.(효율성 증가)
- 목적 파일에 함수가 많다면 사용 여부에 관계없이 모두 실행 파일로 만들어져서 실행 파일 크기가 증가 함
-> C 언어는 사용한 파일만 포함 한다.
- 목적파일을 별도로 프로그래머가 라이브러리 파일을 만들어 함.
- 요즘은 직접 라이브러리를 개발함 (Visual C++ 프로젝트 생성시 라이브러리 파일 만드는 형식 제공) => .lib 파일
- 소스를 공개 하지 않아도 됨, 재사용 향상

2. C 언어 라이브러리

3. 라이브러리 3-1. 라이브러리

C언어에서 라이브러리

- C언어에서는
 - 자주 사용하는 함수들을 미리 작성하여 두고,
 - . 상세 구현 사항은 별도로 모아 저장 시켜 둔 것
 - . 통상, 미리 컴파일 되어 라이브러리 파일 형태로 제공
 - 그 프로토타입(함수 원형) 만을 공개 함

C언어 라이브러리 사용 방법

- 라이브러리 파일
 - C 라이브러리 함수들에 대해 미리 컴파일 된 파일들
- 헤더 파일 (.h)
 - 헤더 파일에는 주로 함수 프로토타입(Function Prototype, 함수 원형)이 들어있게 됨
 - stdio.h 내용

```
int __cdecl fprintf(FILE * __restrict__ _File, const char * __restrict__ _Format, ...);  
int __cdecl printf(const char * __restrict__ _Format, ...);  
int __cdecl sprintf(char * __restrict__ _Dest, const char * __restrict__ _Format, ...)
```

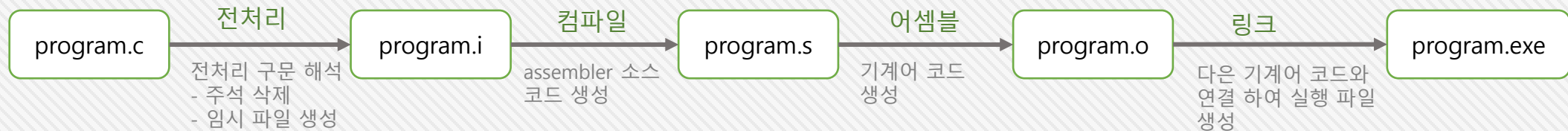
- C 라이브러리의 사용 방법
 - 함수 프로토타입이 있는 헤더 파일을 프로그램 처음에 포함(#include) 시켜 사용

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

```
#include <stdio.h>  
  
int main() {  
    int age;           // 변수 선언  
  
    age = 30;          // 변수 초기화  
  
    printf("%d", age); // 30  
}
```

전처리기

- 컴파일러에게 코드의 특성을 알려주는 키워드
- C언어에서 전처리 구문은 C 컴파일러가 컴파일을 하기전에 전처리기가 선행으로 처리되는 부분
- ; (세미콜론)을 사용 하지 않음
- C언어 컴파일 과정



- > gcc -o program program.c : 임시파일, assembler 파일 삭제 됨
- > gcc -save-temps -o program program.c : 임시파일, assembler 파일 삭제 하지 않음

C언어 전처리기 종류

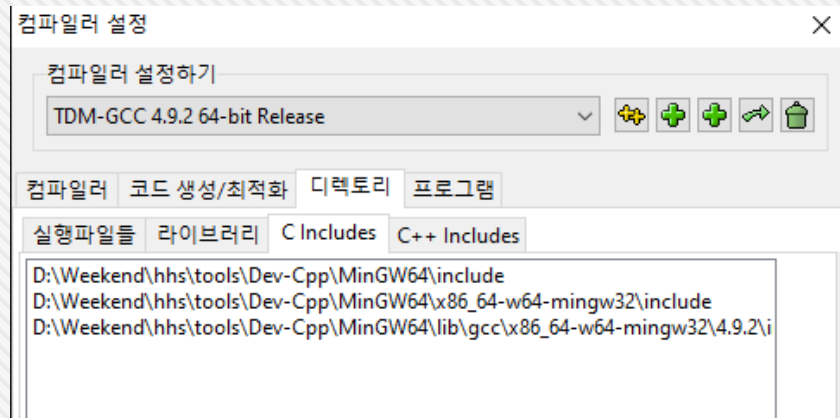
- 파일 처리를 위한 전처리문 : **#include**
- 형태 정의를 위한 전처리문 : **#define, #undef**
- 조건 처리를 위한 전처리문 : **#if, #ifdef, #ifndef, #else, #elif, #endif**
- 에러 처리를 위한 전처리문 : **#error**
- 디버깅을 위한 전처리문 : **#line**
- 컴파일 옵션 처리를 위한 전처리문 : **#pragma**

2. #include 전처리기

3. 라이브러리 3-2. 전처리기

#include 전처리기

- 컴파일러에게 명시한 파일을 읽어오도록 지시 – 다른 소스 파일 (header file) 전체를 치환 즉 복사 함
- 사용법
 - #include "header file" : 현재 디렉토리를 에서 파일을 찾고 없으면 compile option -i로 설정된 directory 또는 표준 header file을 찾음
: #include "myhead,..h" , #include "c:\mydirectory\myhead.h"
 - #include <header file> : C언어 표준 header이거나 Compile option -I디렉터리명 으로 header 파일의 위치를 지정한 디렉토리에 있는 파일만 찾음
: #include <stdio.h>
: 개발 tool을 사용 하는 경우 compile 설정 부분에 있음



- myHeader.h

```
#define BASEVALUE 1
```

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

void main() {
    printf("%d", BASEVALUE);
}
```

3. define 전처리기

3. 라이브러리 3-2. 전처리기

#define

- 상수나 명령문을 치환
- 사용법
 - #define 매크로명
 - #define 매크로상수명 상수값
 - #define 매크로함수명(...) 매크로함수

```
#include <stdio.h>

#define TRUE 1           // 상수 정의
#define FALSE 0
#define MAX_COUNT 3
#define ADD(a, b) (a+b)  // 함수 정의
#define MAX(a, b) (a > b ? TRUE : FALSE)

void main() {
    int maxCnt = MAX_COUNT;
    int value = ADD(3,4);

    printf("%d %d \n", maxCnt, value); // 3, 7
    printf("%d \n", MAX(3,5));          // 0
}
```

```
#define NUMBERS 1, \
                2, \
                3

int x[] = { NUMBERS }; // ==> int x[] = { 1, 2, 3 }; 와 같다.
```

- 주의 사항 : 연산자 우선 순위로 고려 해야 함

```
#include <stdio.h>

#define MULT01(a, b) a * b
#define MULT02(a, b) (a) * (b)
void main() {

    long value01 = MULT01(20 + 4, 7 - 5) * 2;
                    // (20 + 4 * 5 - 6) * 2;
    long value02 = MULT02(20 + 4, 7 - 5) * 2;
                    // ((20 + 4) * (5 - 6)) * 2;
    printf("%ld %ld", value01, value02); // 38, 96

}
```

```
#include <stdio.h>

#define ADD(a, b) printf ("Print Start....\n"); \
    printf ("%d + %d = %d\n", (a), (b), (a)+(b)); \
    printf ("Print End ....");

void main() {
    ADD(3,7);
}
```


4. undef 전처리기

#undef

- 매크로를 정의하고 해제
- 사용법
 - #undef 매크로명

```
#include <stdio.h>

#define TRUE 1
#define FALSE 0
#define MAX_COUNT 3
#define ADD(a, b) (a+b)
#define MAX(a, b) (a > b ? TRUE : FALSE)

void main() {
    int maxCnt = MAX_COUNT;
    int value = ADD(3,4);

    printf("%d %d \n", maxCnt, value);
    printf("%d \n", MAX(3,5));
    #undef MAX
    #define MAX(a, b) (a > b ? FALSE : TRUE)
    printf("%d \n", MAX(3,5));
}
```

5. if 전처리기

3. 라이브러리 3-2. 전처리기

#if

- 매크로 `#if`, `#ifdef`, `#ifndef`, `#else`, `#elif`, `#endif`는 조건부 컴파일하도록 정의
- 조건이 맞지 않으면 컴파일에서 제거됨
 - 즉, 조건이 맞지 않으면 소스 자체가 없는 것과 같은 효과
- 문법

- **#if #elif #else #endif**
: 조건문과 용도가 비슷

```
#include <stdio.h>
#define NUM 10

int main() {
    #if NUM == 10
        printf("매크로로 정의된 NUM의 값은 10\n");
    #elif NUM == 20
        printf("매크로로 정의된 NUM의 값은 20\n");
    #else
        printf("매크로로 정의된 NUM의 값은 10도 20도 아님\n");
    #endif

    return 0;
}
```

- **#ifdef**
: 정의된 매크로에 대한 조건문
: 매크로가 정의되어있는지만에 대해서 조건을 수행

```
#include <stdio.h>
#define NUM 10 // -> 이것이 없으면 ,,,

int main() {
    #ifdef NUM
        printf("매크로로 NUM이 정의되어 있습니다.\n"); //
    #else
        printf("매크로로 NUM이 정의되어 있지않습니다.\n");
    #endif

    return 0;
}
```

- **#if 와 #ifdef 차이점**
: `#if` 는 값이 중요

```
#include <stdio.h>

#define NUM 10

void main() {
    #ifdef NUM
        printf("NUM은 정의된 \n");
    #endif

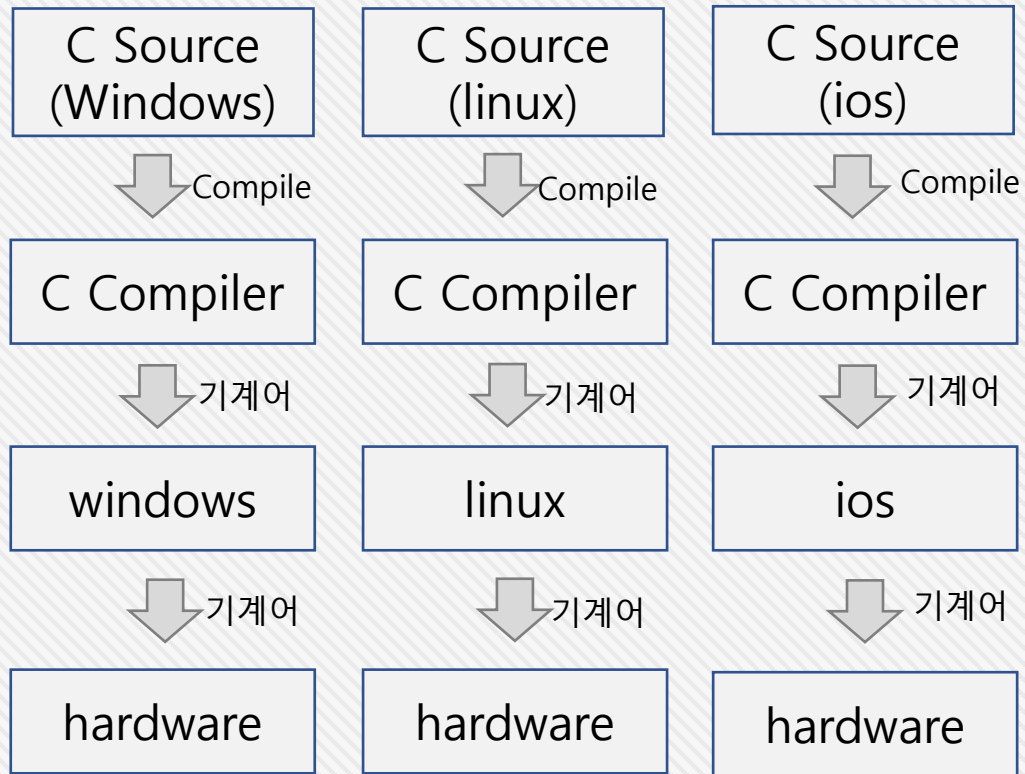
    #if NUM
        printf("NUM은 참 \n");
    #else
        printf("NUM은 거짓 \n");
    #endif
}
```

1. C 표준 라이브러리

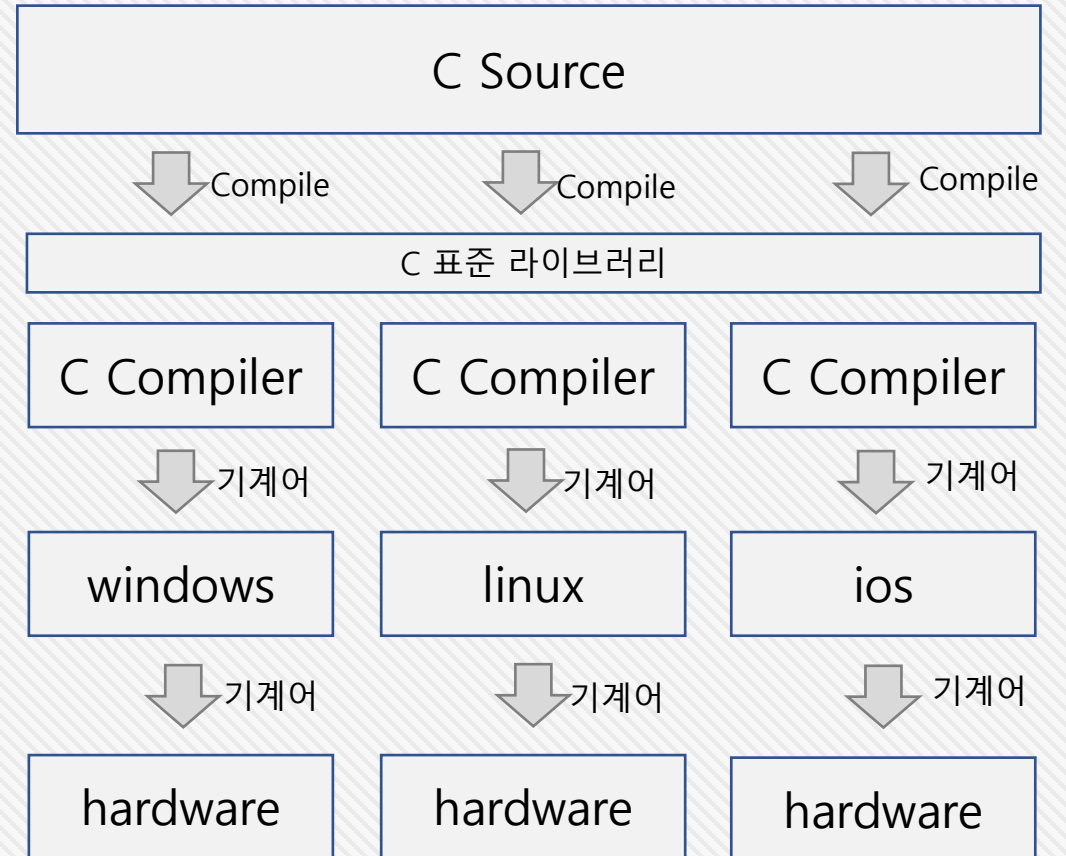
3. 라이브러리 3-3. 표준라이브러리

표준라이브러리 필요 이유

- OS 마다 하드웨어를 관리 하는 방식이 틀리므로 OS에 따라서 틀리게 구현 한다면 매번 소스를 수정 해야 함



- System에 관련된 함수를 제공 하므로 다른 os에서 사용시 소스 수정 없이 사용 가능함



2. C 표준 단일 문자 출력 함수

표준 출력 함수

- 표준 출력 : 시스템이 가장 기본으로 사용하는 출력 방식
 - 출력 : 모니터, 프린터, 스피커 등 ...
 - 컴퓨터의 표준 출력은 모니터
 - stdio.h (Standard Input Output Header)

단일 문자 출력 함수

- putChar : 문자를 출력 한다.
 - 함수 원형 : `int putchar(int character);`
 - 인자 : character
 - 리턴값 : 오류가 하나도 없다면 표준 출력에 쓰여진 문자가 반환된다.
만일 오류가 발생한다면 EOF 가 반환되고 오류 표시자가 설정된다
 - 예제 : 화면에 A를 출력 한다.
- putc : 스트림에 문자를 쓴다.
 - 함수 원형 : `int putc(int character, FILE* stream);`
 - 인자 : character : 스트림에 쓰여질 문자.
stream : 문자가 쓰여질 스트림의 FILE 객체를 가리키는 포인터
 - 리턴값 : 오류가 하나도 없다면 스트림에 쓰인 문자가 똑같이 반환된다.
오류가 발생한다면 EOF 가 리턴되고 오류 표시자가 설정된다.
 - 예제 : 화면에 a를 출력 한다.
 - file에 write 하는 것은 file open에서 학습

```
#include <stdio.h>

void main() {
    char ch = 'A';      /* 문자 상수 */
    char chAscii = 65;
    putchar(chAscii);   /* 아스키코드 65는 문자 A 로 A 출력됨 */
    putchar(ch);         /* 문자 상수 A 출력됨 */
}
```

```
/* stdout (표준 출력) 에 a 를 쓴다.*/
#include <stdio.h>
int main() {
    char ch = 'a';

    putc(ch, stdout);

    return 0;
}
```

2. C 표준 문자열 출력 함수 – puts, printf

3. 라이브러리
3-4. 표준입출력함수

문자열 출력 함수

- **puts** : 표준 출력에 문자열을 쓴다.

- 함수 원형 : **int puts(const char* str);**
- 인자 : str: 표준 출력에 쓰여질 C 형식 문자열
- 리턴값 : 성공적으로 쓰였다면 음이 아닌 값이 리턴 된다.
오류가 발생하였다면 EOF 를 리턴 한다.
- 설명 : 문자열에 \n 이 마지막에 없더라도 한 줄 개행이 되어 출력
- 예제 : "Hello, C World!" 출력

```
#include <stdio.h>
int main() {
    puts("Hello, C World! \n"); /* \n이 있는 것과 없는 것 차이점 확인 */

    char str[] = "Hello, C World!";
    puts(str);

    return 0;
}
```

- **printf** : 표준 출력에 문자열을 쓴다.

- 함수 원형 : **int printf(const char* format, ...);**
- 인자 : format : 형식 문자열(format:서식지정자)형식 문자열
... :
- 리턴값 : 출력에 성공하면 출력된 전체 문자의 개수가 리턴된다. 출력에 실패하면 음수가 리턴
- 설명 : 표준 출력(stdout) 에 일련의 데이터들을 형식 문자열(format)에 지정되어 있는 형태로 출력
puts와 틀리게 한 줄 개행이 되지 않음

```
#include <stdio.h>

int main() {
    printf("Hello, C World!\n");
    printf("%s", "Hello, C World!");

    return 0;
}
```

3. 형식문자열(서식지정자)

형식문자열(서식지정자)

• 문법 : %[플래그(flag)][폭(width)][정밀도][크기(length)]서식 문자(specifier)

- 플래그(flag) : 기본적으로 출력되는 형태에 대해 조금 더 자세하게 지정할 수 있게 해준다

플래그	설명
+	출력 결과값이 양수인 경우라도 + 기호를 앞에 붙여서 출력하도록 한다. (물론 음수면 자동적으로 - 가 붙는다). 기본적으로 지정하지 않았을 경우 음수에만 앞에 - 가 붙는다.
(공백)	앞에 부호가 붙지 않는다면 한 칸을 띄워서 출력한다. (123 은 " 123" 으로 출력되고 -123 은 "-123" 으로 출력된다)
#	o, x, X 서식 문자들과 사용되면 출력되는 값 앞에 각각 0, 0x, 0X 가 붙게 된다. (이 때 0 은 제외한다). e, E, f 서식 문자들과 사용되면 소수점 아래 수들이 없음에도 불구하고 강제적으로 소수점을 붙이도록 한다. 원래 소수점 아래 수들이 없다면 소수점을 붙이지 않는다. g 와 G 서식 문자들과 사용되면 e 와 E 일때와 동일한 작업을 하지만 소수들의 뒷부분에 붙는 0 들 (123.1200 등) 은 제거되지 않는다.
0	수들을 왼쪽으로 정렬하되 빈 칸을 삽입하는 대신에 0 을 삽입한다.

- 폭(width) : 폭은 말그대로 출력되는 데이터의 폭을 지정

폭	설명
*	폭을 형식 문자열에 지정해서 받지 않지만, 그 대신에 형식 문자열 뒤에 오는 인자들에 넣어서 받는다. 이 때, 이는 정수 값이어야 하며 폭을 지정하는 변수 뒤에 출력할 데이터가 위치하면 된다.

3. 형식문자열(서식지정자)

형식문자열(서식지정자)

• 문법 : %[플래그(flag)][폭(width)][.정밀도][크기(length)]서식 문자(specifier)

- 정밀도 : 말그대로 수치 데이터를 출력할 때 어떠한 정밀도로 출력하는지 (즉, 몇 자리 까지 출력해야 되는지) 를 지정
앞에 마침표(.) 사용 => 폭과 구분을 하기 위해서

정밀도	설명
.*	형식 문자열에서 정밀도를 나타내지는 않지만 뒤에 인자로 정밀도 값을 준다. 이 때 인자는 형식 태그가 적용되는 데이터 앞에 있어야 한다.

- 크기(length) : 데이터의 정확한 크기를 지정하는데 사용.
예를 들어서 %d 서식문자의 경우 막연하게 '정수형 데이터를 십진법으로 출력한다' 였지만
길이를 지정해주면 어떻게 크기로 데이터를 출력 해야 되는지 (int 나 short 나 등등) 을 지정할 수 있음

길이	설명
l	정수 서식 문자(i,d,o,u,x, X) 에 사용되었을 경우 인자를 long int 나 unsigned long int c 나 s 에 사용되었을 경우 wide character 나 wide string 으로 생각한다.
L	인자를 long double 로 생각한다. (오직 부동 소수점 서식 문자인 e,E,f,g, G 에만 적용된다)

3. 형식문자열(서식지정자)

형식문자열(서식지정자)

• 문법 : %[플래그(flag)][폭(width)][정밀도][크기(length)]서식 문자(specifier)

- 서식문자 : 말그대로 수치 데이터를 출력할 때 어떠한 정밀도로 출력하는지 (즉, 몇 자리 까지 출력해야 되는지) 를 지정
앞에 마침표(.) 사용 => 폭과 구분을 하기 위해서

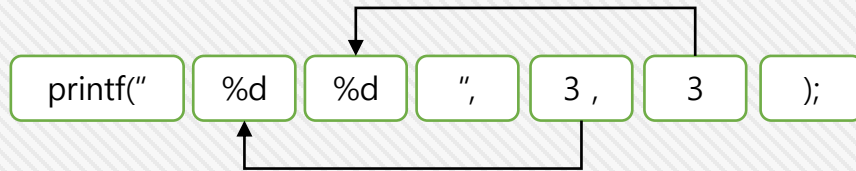
서식 문자	출력 형태	예시
c	문자	a
d or i	부호 있는 십진법으로 나타난 정수	392
e	지수 표기법(Scientific notation) 으로 출력하 되, e 문자를 이용.	3.9265e+2
E	지수 표기법(Scientific notation) 으로 출력하 되, E 문자를 이용.	3.9265E+2
f	십진법으로 나타난 부동 소수점 수	392.65
g	%e나 %f 보다 간략하게 출력	392.65
G	%E나 %f 보다 간략하게 출력	392.65
o	부호 있는 팔진수	610

서식 문자	출력 형태	예시
s	문자열	sample
u	부호없는 십진법으로 나타난 정수	7235
x	부호없는 16 진법으로 나타난 정수 (소문자 사용)	7fa
X	부호없는 16 진법으로 나타난 정수 (대문자 사용)	7FA
p	포인터 주소	B800:0000
n	아무것도 출력하지 않는다. 그 대신, 인자로 부호 있 는 int 형을 가리키는 포인터를 전달해야 되는데, 여 기에 현재까지 쓰여진 문자 수가 저장된다.	
%	% 다음에 %를 또 붙이면 stdout 에 % 를 출력한다.	

4. printf

3. 라이브러리 3-4. 표준입출력함수

printf 사용 방법



%c : 변수가 가지고 있는 값을 ASCII 표에 대응 하는 문자 출력

```
#include <stdio.h>

void main() {
    char c = 65;
    printf("65의 ASCII 값은%c 입니다.\n", 65);
    printf("65의 ASCII 값은%c 입니다.\n", c);
}
```

변수

Locals

c: 0 '\000'

> Registers

변수

Locals

c: 65 'A'

> Registers

65의 ASCII 값은A 입니다.
65의 ASCII 값은A 입니다.

%f : 실수형 출력

```
#include <stdio.h>

void main() {
    float fa = 3.3f; /* float 형식 */
    float fb = 3.3; /* double 형식 */
    printf("fa %f.\n", fa);
    printf("fb %f.\n", fb);
    printf("fa %d.\n", fa); /* 잘못된 type 지정시 이상한 값*/
}
```

변수

Locals

fa: 0

fb: 2.24207754e-44

> Registers

변수

Locals

fa: 3.29999995

fb: 3.29999995

> Registers

fa 3.300000.
fb 3.300000.
fa 1610612736.

4. printf

%u : 부호 없는 10진수 출력

- unsigned int : 부호 있는 4byte (부호 1bit, 테이터 31bit : 범위) : 범위 -2.147.483.648 ~ 2.147.483.647)
- signed int : 부호 없는 4Byte (테이터 32bit : 범위) 범위 : 0 ~ 4,294.967.295)

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

```
void main() {
```

```
    int num01 = -1; /* 최대값 */
```

```
    unsigned int num02 = 4294967295;
```

```
    printf("부호고려 num01 : %d, 부호고려하지않음 num01 : %u \n", num01, num01);
```

```
    printf("부호고려 num02 : %d, 부호고려하지않음 num02 : %u \n", num02, num02);
```

```
}
```

```
부호고려 num01 : -1, 부호고려하지않음 num01 : 4294967295
부호고려 num02 : -1, 부호고려하지않음 num02 : 4294967295
```

- %d, %u는 출력 크기를 4Byte로 변환 하여 출력 – char 에 음수 입력 시 주의

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

```
void main() {
```

```
    char ch01 = -1;
```

```
    char ch02 = 1;
```

```
    char ch03 = 'A';
```

```
    printf("부호고려 ch : %d, 부호고려하지않음 ch : %u \n", ch01, ch01);
```

```
    printf("부호고려 ch : %d, 부호고려하지않음 ch : %u \n", ch02, ch02);
```

```
    printf("부호고려 ch : %d, 부호고려하지않음 ch : %u \n", ch03, ch03);
```

```
}
```

```
부호고려 ch : -1, 부호고려하지않음 ch : 4294967295
부호고려 ch : 1, 부호고려하지않음 ch : 1
부호고려 ch : 65, 부호고려하지않음 ch : 65
```

4. printf

%o 8진수 %x 16진수 출력 (%X 대문자 출력) 형태의 정수 출력

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

```
void main() {  
    int num = 10;  
    int octal = 010;  
    int hexa = 0x10;  
    printf("num    => 10진수 num : %d, 8진수 변환 : %o, 16진수 변환 : %X \n", num, num, num);  
    printf("octal  => 10진수 num : %d, 8진수 변환 : %o, 16진수 변환 : %X \n", octal, octal, octal);  
    printf("hexa   => 10진수 num : %d, 8진수 변환 : %o, 16진수 변환 : %X \n", hexa, hexa, hexa);  
}
```

```
num    => 10진수 num : 10, 8진수 변환 : 12, 16진수 변환 : A  
octal  => 10진수 num : 8, 8진수 변환 : 10, 16진수 변환 : 8  
hexa   => 10진수 num : 16, 8진수 변환 : 20, 16진수 변환 : 10
```

%e 실수를 지수로 출력

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

```
void main() {  
    float f_num = 22.57f;  
  
    printf("f_num    => %f , %e, %E \n", f_num, f_num, f_num);  
}
```

```
f_num    => 22.570000 , 2.257000e+001, 2.257000E+001
```

4. printf

출력 칸 수 조절

- %출력칸수d
- 오른쪽 정렬 : 양수, 왼쪽 정렬 : 왼쪽

```
#include <stdio.h>

int main()
{
    int num = 10;
    printf("[%d], [%5d], [%-5d]", num, num, num);
    return 0;
}
```

[10], [_ 10], [10 _]

실수의 소수점 자리

- % + 전체 칸 수 + . + 소수점 자리수 + f

```
#include <stdio.h>

int main()
{
    double num = 1.02345;
    printf("[%f], [%.4f], [%10.5f], [%-10.5f], [%15.10f], ", num, num, num, num, num);
    return 0;
}
```

[1.023450], [1.0235], [1.02345], [1.02345 _], [1.0234500000],

제어 코드

- 소리를 표시 하거나 콘솔의 출력과 입력의 위치를 변경 하는 코드

제어 코드	기능
\n	줄바꿈
\r	해당 줄의 처음으로 이동
\t	tab (코드를 이쁘게 정렬할 때 사용)
\b	바로 앞칸으로 이동
\a	시스템 스피커로 경고음 발생
\'	작은 따옴표 출력
\"	큰 따옴표 출력
\\	\ 출력

```
#include <stdio.h>

int main()
{
    printf("안녕하세요\n");
    printf("대한 민국 역사는\r만년이다.\n");
    printf("대한 민국 역사는\t만년이다.\n");
    printf("대한 민국 역사는\b만년이다.\n");
    printf("대한 민국 역사는\a만년이다.\n");
    printf("대한 민국 역사는\'만년\'이다.\n");
    printf("대한 민국 역사는\"만년\"이다.\n");
    printf("대한 민국 역사는\\만년이다.\n");
}
```

```
C:\WINDOWS\system32\cmd.exe

안녕하세요
만년이다.역사는
대한 민국 역사는      만년이다.
대한 민국 역사는    만년이다.
대한 민국 역사는  만년이다.
대한 민국 역사는'만년'이다.
대한 민국 역사는"만년"이다.
대한 민국 역사는\\만년이다.

Press any key to continue . . .
```

6. C 표준 입력 함수

3. 라이브러리
3-4. 표준입출력함수

표준 입력 함수

- 표준 출력 : 시스템이 가장 기본으로 사용하는 입력 방식
 - 출력 : 키보드, 스캐너 ...
 - 컴퓨터의 표준 출력은 키보드
 - stdio.h (Standard Input Output Header)

사용자 입력
"abc"
(키보드)

표준 입력
버퍼
a b c

엔터

컴퓨터 사용

- 문자를 한 개만 받는 입력 함수를 사용 한 경우

사용자 입력
"abc"
(키보드)

표준 입력
버퍼
a b c

엔터

표준 입력
함수 a 사용
a b c

표준 입력
함수 b 사용
b c

- Buffer 에 남아 있는 b 사용
- rewind() 함수를 이용해서 초기화 필요

6. C 표준 단일 입력 함수

단일 입력 함수

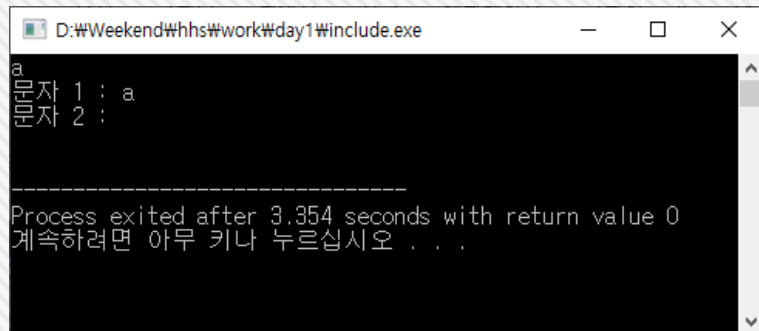
- **getchar** : stdin 에서 한 문자를 가져온다.
 - 함수 원형 : **int getchar(void);**
 - 인자 : 없음
 - 리턴값 : 읽어드린 문자를 int 값으로 리턴.
 - 설명 : unsigned char 로 받은 문자를 int 로 변환해서 리턴.
오류 발생시에 EOF 를 리턴
 - 예제 : 문자 하나를 입력 받아서 출력 한다,

```
#include <stdio.h>
int main() {
    int ch = getchar();
    printf("문자 : %c \n", ch);

    return 0;
}
```

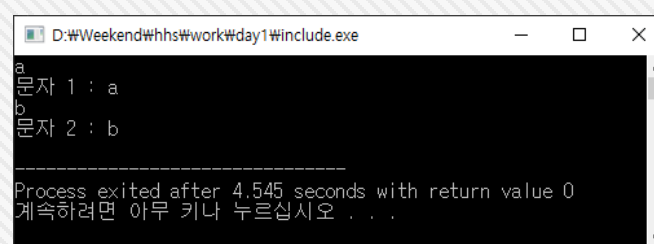
* enter(10 , \n)을 입력 받게됨 => 빈 공간

```
#include <stdio.h>
int main() {
    int ch = getchar();
    printf("문자 1 : %c \n", ch);
    ch = getchar();
    printf("문자 2 : %c \n", ch);
    return 0;
}
```



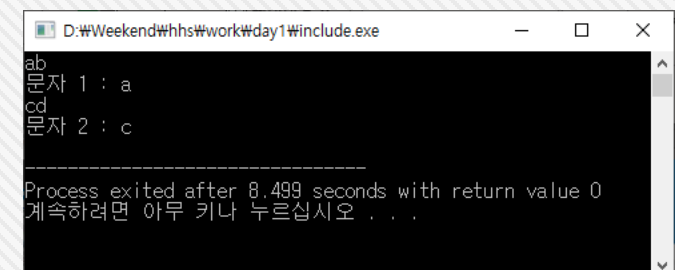
* 한번 호출을 더 해서 엔터 제거

```
#include <stdio.h>
int main() {
    int ch = getchar();
    printf("문자 1 : %c \n", ch);
    ch = getchar();
    printf("문자 2 : %c \n", ch);
    return 0;
}
```



* 입력버퍼제거

```
#include <stdio.h>
int main() {
    int ch = getchar();
    rewind(stdin);
    printf("문자 1 : %c \n", ch);
    ch = getchar();
    rewind(stdin);
    printf("문자 2 : %c \n", ch);
    return 0;
}
```



6. C 표준 단일 입력 함수

단일 입력 함수

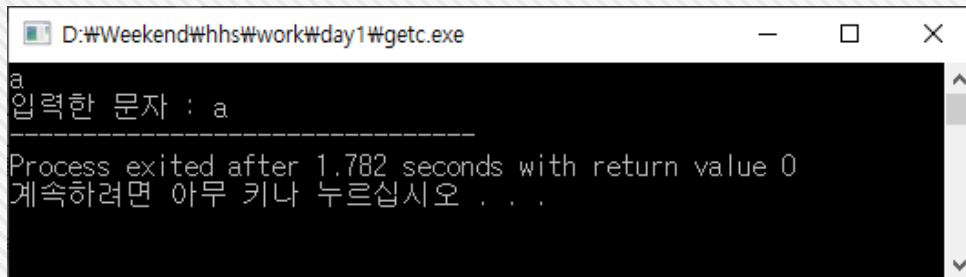
- **getc** : 스트림에서 한 문자를 읽어온다.
 - **함수 원형** : `int getc(FILE* stream);`
 - **인자** : 문자를 읽어올 스트림의 FILE 객체를 가리키는 포인터
 - **리턴값** : 읽어드린 문자를 int 값으로 리턴.
읽기 오류가 발생한다면 함수는 EOF 를 리턴하고 이에 대응하는 오류 혹은 EOF 표시자가 설정
 - **설명** : 문자를 읽어온 스트림의 내부 파일 위치 표시자가 현재 가리키는 문자를 리턴.
그리고 내부 파일 표시자는 그 다음 문자를 가리키게 된다.
 - **예제** : 문자 하나를 입력 받아서 출력 한다,

```
#include <stdio.h>
int main() {
    int c;

    c = getc(stdin);

    printf("입력한 문자 : %c", c);

    return 0;
}
```

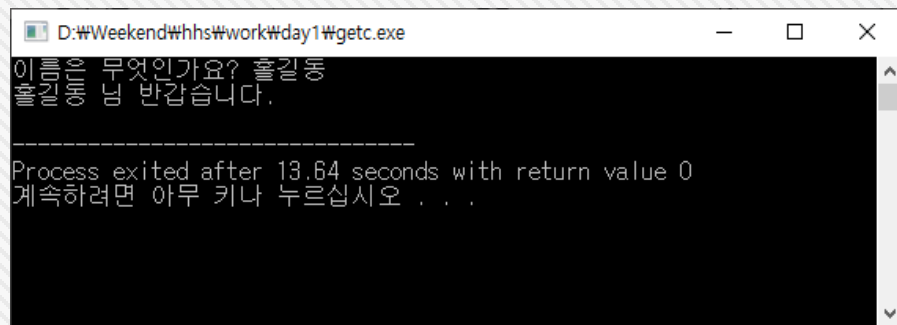


7. C 표준 문자열 입력 함수

문자열 입력 함수

- **gets** : 표준 입력(stdin) 에서 문자열을 가져온다.
 - **함수 원형** : `char* gets(char* str);`
 - 인자 : 문자를 읽어올 스트림의 FILE 객체를 가리키는 포인터
 - 리턴값 : 성공적으로 읽어 들였다면 str 을 리턴
오류가 발생했다면 NULL 포인터가 리턴
 - 설명 : 표준 입력에서 문자들을 개행 문자 ('\n') 이나 파일 끝(Eof) 를 만나기 전 까지 가져와서 str 에 저장.
이 때, 개행 문자 ('\n') 은 문자열에 포함되지 않는다. 널 문자 ('\0') 는 문자열 맨 마지막에 자동적으로 추가된다..
 - 예제 : 이름을 입력 받아서 출력
 - 주의사항 : 최근의 C 표준 (2011) 에서 이 함수는 라이브러리에서 삭제됨,

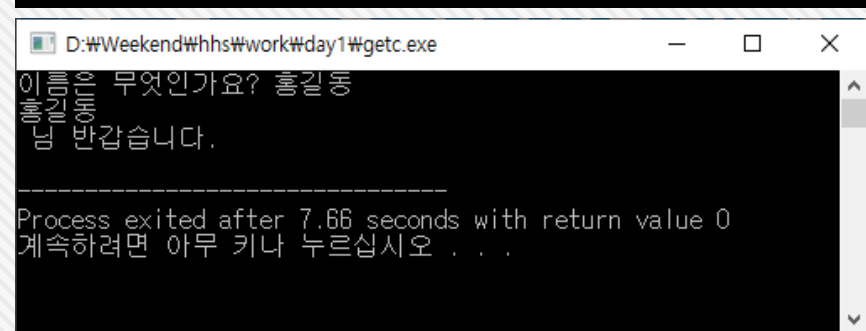
```
#include <stdio.h>
int main() {
    char str[100];
    printf("이름은 무엇인가요? ");
    gets(str);
    printf("%s 님 반갑습니다.\n", str);
    return 0;
}
```



```
D:\Weekend\hhs\work\day1\getc.exe
이름은 무엇인가요? 홍길동
홍길동 님 반갑습니다.

Process exited after 13.64 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```

```
#include <stdio.h>
int main() {
    char str[100];
    printf("이름은 무엇인가요? ");
    fgets(str,100, stdin);
    printf("%s 님 반갑습니다.\n", str);
    return 0;
}
```



```
D:\Weekend\hhs\work\day1\getc.exe
이름은 무엇인가요? 홍길동
홍길동
 님 반갑습니다.

Process exited after 7.66 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```

7. C 표준 문자열 입력 함수

문자열 입력 함수

- **scanf** : 표준입력(stdin) 으로 부터 데이터를 형식에 맞추어 읽어온다.
 - 함수 원형 : **int scanf(const char* format, ...);**
 - 인자 : 형식(format) 문자열
 - 리턴값 : 입력이 성공적이였다면 함수는 성공적으로 읽어들이 인자의 개수를 리턴
 - 설명 : 표준입력(stdin) 으로 부터 데이터를 읽어와 형식(format) 문자열에 따라 나머지 인자들이 가리키는 장소에 값을 대입한다.
이 때, 나머지 인자들은 반드시 할당된 공간을 가리켜야 하며, 형식 문자열의 형식 태그(format tag) 가 지정하는 바에 따라 대응되는 인자들이 가리키는 공간에 값이 대입된다
 - 예제 :

형식(format) 문자열

- **공백 문자(Whitespace character)**
: scanf 함수는 공백문자(띄어쓰기 한 칸, 개행 문자, 탭 문자)가 아닌 것들 이전에 나오는 모든 공백 문자 를 모두 무시한다.
- **퍼센트 기호(%) 를 제외한 비-공백문자들(non-whitespace character)**
: 형식 문자열에 있는 공백 문자나 형식 지정자(% 로 시작하는 것들) 을 제외한 나머지 문자들은 scanf 함수로 하여금 stdin 에서 다음 문자를 읽은 후 해당 비-공백 문자와 비교하여 같다면 무시한 후, 형식 문자열의 다음 문자들을 처리하고, 다르다면 함수를 종료하게 되고 stdin 에는 읽히지 않은 다음 문자들이 남아 있게 된다.
- **형식 지정자(Format specifier)**
: % 다음에 오는 문자들은 scanf 함수의 형식 지정자를 나타내며 이 형식 지정자는 stdin 에서 어떠한 타입과 형식의 데이터를 가져올지에 대해서 알려준다.
이 때, 형식 지정자에 따라 stdin 에서 입력받은 데이터는 각 형식 지정자에 대응되는 인자들이 가리키는 주소에 저장된다

7. C 표준 문자열 입력 함수

형식 지정자(Format specifier)

- %[*][폭(width)][한정자(modifiers)]타입(type)

종류	설명
*	데이터를 stdin 에서 받아들이지만 무시된다. 물론, 이에 대응되는 인자에는 받아들인 데이터가 저장되지 않고 이 인자는 다음 형식 태그에 대응된다. 예를 들어 scanf("%*d%d", i, j); 의 경우 먼저 수를 입력하더라도 %*d 형식이므로 무시 된다. 그 다음 수를 입력하면 %d 형식 태그가 j 가 아닌 i 에 대응되어 i 에 그 다음 입력한 수가 들어가게 된다. 이 때 j 에는 아무런 값도 들어가지 않는다.
폭	stdin 에서 읽어들이 최대 문자 수를 지정한다. 예를 들어 scanf("%10s", str); 로 했을 경우 stdin 에서 최대 10 문자를 읽어와 str 에 저장한다. 이 때 주의할 점은 str 에는 NULL 문자가 들어갈 수 있는 충분한 공간이 남아 있어야 한다.
한정자	입력받는 데이터의 크기를 지정한다. int, unsigned int, float 형에 대해 입력받는 데이터의 크기를 설정할 수 있다. h 의 경우 short int (d, i, n 의 경우) 혹은 unsigned short int (o, u, x 일 경우). l 의 경우 long int (d, i, n 의 경우) 혹은 unsigned long int (o, u, x 일 경우), 혹은 double (e, f, g 일 경우). 마지막으로 L 의 경우 long double (e, f, g 일 경우) 에 사용할 수 있다.
타입	데이터를 어떠한 형식으로 혹은 어떠한 값만을 읽어들이어야 할 지에 대해 지정해준다. 아래 표를 참고.

7. C 표준 문자열 입력 함수

scanf 함수의 타입 지정자들

타입	대응되는 입력 방식	대응되는 인자의 형태
d	십진법으로 표현된 정수: 말그대로 십진법으로 쓰인 정수로, + 나 - 기호로 시작할 수도 있다.	int *
e, E, f, g, G	부동 소수점: 소수점을 포함하고 있는 소수(decimal number) 로 + 나 - 기호로 시작할 수도 있으며, e 나 E 문자(10의 지수를 나타내기 위해)를 포함할 수 도 있다. -732.103, 12e-4, +123.10 은 모두 올바른 입력이다.	float *
o	8진법으로 표현된 정수	int *
s	문자열: 공백문자를 찾을 때 까지 문자들을 읽어들인다.	char *
u	부호가 없는 십진법으로 표현된 정수	unsigned int *
x, X	16진법으로 표현된 정수	int *

7. C 표준 문자열 입력 함수

3. 라이브러리
3-4. 표준입출력함수

예제

```
#include <stdio.h>

int main() {
    char str[10];
    char ch;
    int dec, hex, oct;
    float db;
    printf("문자열, 문자, 십진수, 16 진수, 8 진수, 소수를 각각 입력하세요\n");
    scanf("%9s %c %d %x %o %f", str, &ch, &dec, &hex, &oct, &db);
    printf("문자열 : %s\n", str);
    printf("문자 : %c\n", ch);
    printf("십진수 : %d\n", dec);
    printf("16 진수 : %x\n", hex);
    printf("8 진수 : %o\n", oct);
    printf("소수 : %f\n", db);
    return 0;
}
```

```
문자열, 문자, 십진수, 16 진수, 8 진수, 소수를 각각 입력하세요
String
C
10
2F
10
2.5
문자열 : String
문자 : 1
십진수 : 0
16 진수 : 2f
8 진수 : 10
소수 : 2.500000
```

Content

IV. 프로그램 제어

1. 연산자
2. 순서도
3. 제어문
4. 반복문
5. 비트연산
6. 시프트연산자
7. Bit 연산자
8. 변수 범위

연산자

- 주어진 식을 계산하여 결과를 얻는 과정을 연산이라고 하며, 연산을 수행 하는 기호
- 피연산자 : 연산자의 작업 대상(변수, 상식, 수식)

종류	연산자
대입 연산자	=
산술 연산자	+, -, *, /, &, ++, --
관계 연산자	<, >, <=, >=, ==, !=
논리 연산자	&&, , !
할당 연산자	+=, -=, *=, /=, %=
삼항 연산자	?
비트 연산자	&, , ~, ^, <<, >>

2. 대입 연산자

4. 프로그램 제어

4-1. 연산자

대입연산자

- = 로 나타내며 변수에 상수 값 또는 다른 변수 값을 대입



```
#include <stdio.h>

void main() {
    int data1, data2;
    data1 = 2;
    data2 = 3;
    printf("data1 : %d, data2 : %d\n", data1, data2);
}
```

```
C:\WINDOWS\system32\cmd.exe
data1 : 2, data2 : 3
Press any key to continue . . .
```


3. 산술 연산자

4. 프로그램 제어

4-1. 연산자

산술 연산자

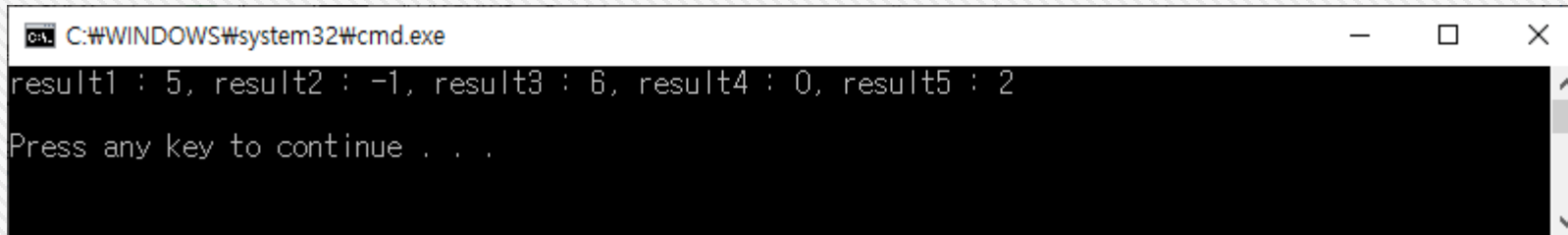
- 상수 또는 변수의 값을 이용 하여 +(더하기), -(빼기), *(곱하기), %(나누기) 함
- 나누기에서 몫과 나머지가 있는데 몫은 / 연산자, 나머지는 % 연산자 사용

```
#include <stdio.h>

void main() {
    int data1, data2;
    data1 = 2;
    data2 = 3;

    int result1 = data1 + data2;
    int result2 = data1 - data2;
    int result3 = data1 * data2;
    int result4 = data1 / data2; /* 2를 3으로 나눔 : 몫을 result4 에 저장 */
    int result5 = data1 % data2; /* 2를 3으로 나눔 : 나머지 result5 에 저장 */

    printf("result1 : %d, result2 : %d, result3 : %d, result4 : %d, result5 : %d\n"
           , result1, result2, result3, result4, result5);
}
```



The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The output of the program is displayed as follows:

```
result1 : 5, result2 : -1, result3 : 6, result4 : 0, result5 : 2
Press any key to continue . . .
```

3. 산술 연산자

4. 프로그램 제어

4-1. 연산자

산술 연산자 - 나머지 연산의 주의 사항

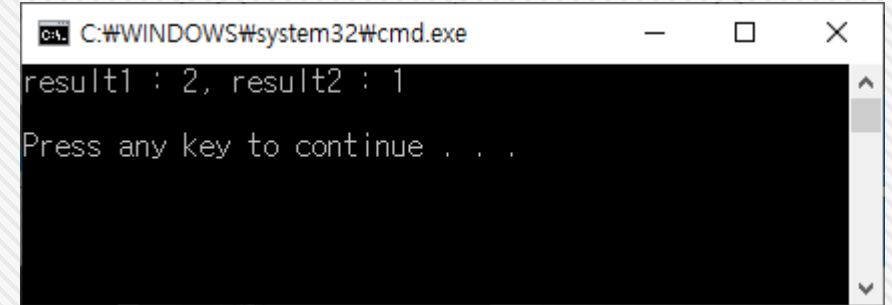
- C언어에서는 정수 끼리 나눗셈을 하면 정수

```
#include <stdio.h>

void main() {
    int data1, data2;
    data1 = 5;
    data2 = 2;

    int result1 = data1 / data2; /* 5를 2으로 나눔 : 몫 */
    int result2 = data1 % data2; /* 5를 2으로 나눔 : 나머지 */

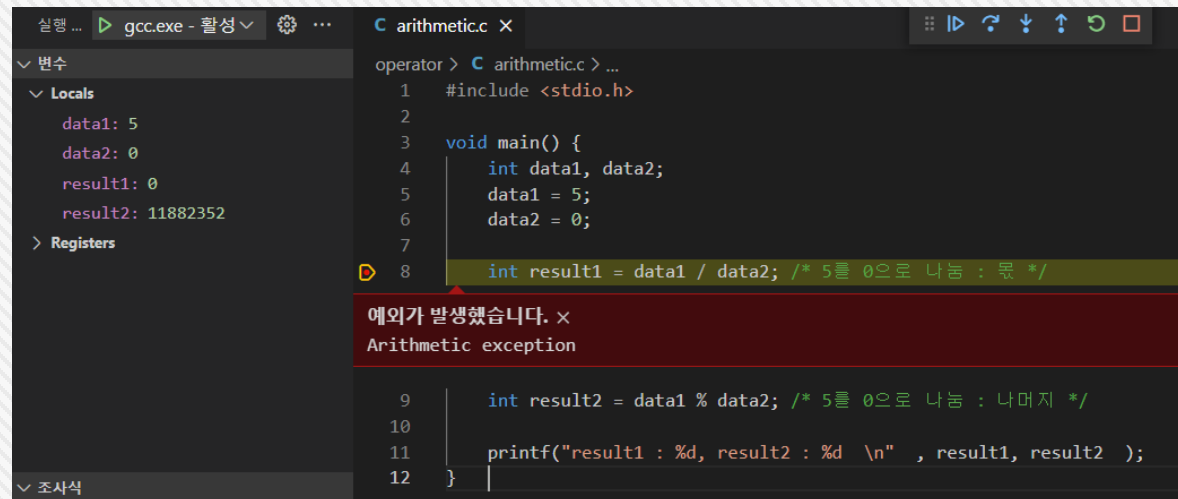
    printf("result1 : %d, result2 : %d \n" , result1, result2 );
}
```



```
C:\WINDOWS\system32\cmd.exe
result1 : 2, result2 : 1
Press any key to continue . . .
```

5/2 는 2.1

- 0으로 나눈다면 어떤 결과 일까 ?
 - : 컴파일시 에러 발생 하지 않음
 - : 실행 시 오류 발생



```
operator > C:\arithmetic.c > ...
1 #include <stdio.h>
2
3 void main() {
4     int data1, data2;
5     data1 = 5;
6     data2 = 0;
7
8     int result1 = data1 / data2; /* 5를 0으로 나눔 : 몫 */
9
10    int result2 = data1 % data2; /* 5를 0으로 나눔 : 나머지 */
11    printf("result1 : %d, result2 : %d \n" , result1, result2 );
12 }
```

예외가 발생했습니다. x
Arithmetic exception

3. 산술 연산자

4. 프로그램 제어

4-1. 연산자

산술 연산자 - 실수

```
#include <stdio.h>

int main()
{
    float num1;
    float num2;

    num1 = 1.23f * 2.58f;    // 1.23을 2.58f를 곱
    num2 = 5.0f / 2.0f;      // 5.0에서 2.0을 나누기

    printf("%f\n", num1);    // 3.173400
    printf("%f\n", num2);    // 2.500000

    return 0;
}
```

```
#include <stdio.h>

int main()
{
    float num1;

    num1 = 5.0f / 0.0f;      // 5.0에서 0.0을 나누기
    printf("%f\n", num1);    // 1.#INF00

    return 0;
}
```

실수에서 0으로 나누면 무한대가 나온다.

0을 어떤 수로 나누면 어떻게 되나요?

- 0 / 10과 같이 0을 10으로 나누면 결과는 0입니다. 즉, $0 / 10 = 0$ 에서 10을 등호 오른쪽으로 보내면 $0 = 0 * 10$ 이 되므로 올바른 식.

나머지 연산은 정수에서만 사용 가능하고 실수에서는 사용할 수 없음

```
18
19 // 나머지 연산
20 float result5 = num1 % 2.0f;

⊗ 92_arithmeticdivOp.c 문제 2개 중 1개
식에 정수 계열 형식이 있어야 합니다. C/C++(31)

21 float result6 = num2 % 2;
```

https://github.com/hyomee/c_basic/blob/main/src/operator/92_arithmeticfloatOp.c

https://github.com/hyomee/c_basic/blob/main/src/operator/92_arithmeticdivOp.c

3. 산술 연산자

4. 프로그램 제어

4-1. 연산자

산술 연산자 - 실수에서 나머지 연산

- math.h 헤더 파일의 fmod, fmodf, fmodl 함수 사용
- fmod : double, fmodf : float, fmodl : long

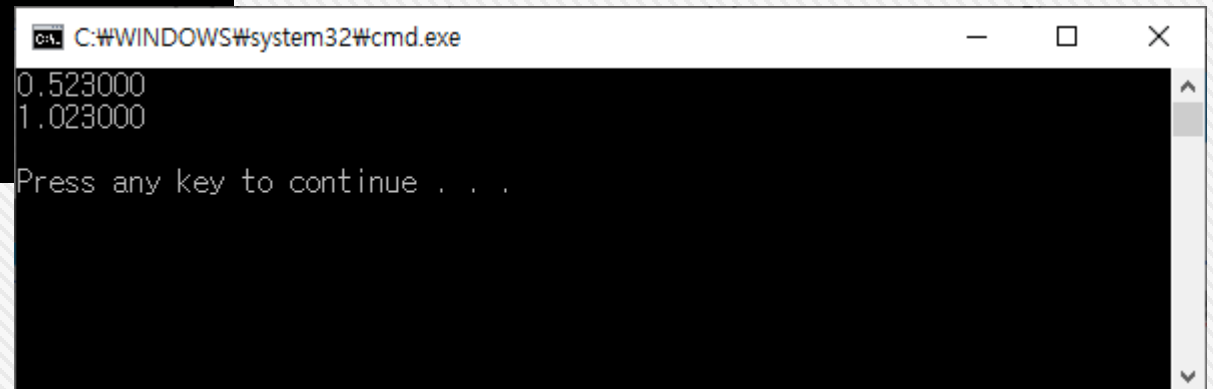
```
#include <stdio.h>
#include <math.h>    // fmod 함수가 선언된 헤더 파일

int main()
{
    // 실수의 나머지 연산은 fmod, fmodf, fmodl 함수를 사용

    double num1 = 5.523;
    double num2 = 1.25;
    printf("%f\n", fmod(num1, num2));    // 0.523000

    float num3 = 5.523f;
    float num4 = 2.25f;
    printf("%f\n", fmodf(num3, num4));    // 1.023000

    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
0.523000
1.023000
Press any key to continue . . .
```

4. 증감 연산자

4. 프로그램 제어 4-1. 연산자

증감 연산자 - ++, --

- ++ : 순차적으로 1씩 증가, -- : 순차적으로 1씩 감소
- 전위형 : 연산을 먼저 수행, 후위형 : 연산을 나중에 수행

The screenshot shows a C program in a code editor with a variable watch window. The code defines variables `i`, `j`, `numPostfix`, and `numPrefix`, all initialized to 10. It then performs `numPostfix = ++i;` and `numPrefix = j--;`. The output shows `numPostfix` is 11 and `numPrefix` is 10.

```
operator > C math.c > main()
1  #include <stdio.h>
2
3  int main() {
4      int i = 10, numPostfix;
5      int j = 10, numPrefix;
6      numPostfix = ++i;
7      numPrefix = j--;
8
9      printf("PostFix : %d, PreFix : %d\n", numPostfix, numPrefix);
10     return 0;
11 }
```

Locals:

- i: 11
- numPostfix: 11
- j: 11
- numPrefix: 10

Registers:

C:\WINDOWS\system32\cmd.exe

```
PostFix : 11, PreFix : 10
Press any key to continue . . .
```

5. 관계 연산자

4. 프로그램 제어

4-1. 연산자

관계 연산자

- 두 수를 비교 하여 결과를 참(1), 거짓(0)으로 표시

```
#include <stdio.h>

int main() {
    int numa = 4, numb = 7;

    int result1 = numa > numb;
    int result2 = numa < numb;
    int result3 = numa == numb;
    int result4 = numa != numb;

    printf(" numa > numb : %d \n", result1);
    printf(" numa < numb : %d \n", result2);
    printf(" numa == numb : %d \n", result3);
    printf(" numa != numb : %d \n", result4);

    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
numa > numb : 0
numa < numb : 1
numa == numb : 0
numa != numb : 1
Press any key to continue . . .
```

6. 논리 연산자

논리 연산자

- 두 값을 해당 하는 논리로 연산
- && : 두 값이 모두 참 이여야 결과 참
- || : 두 값 중 하나라도 참이면 결과 참
- ! : 값이 거짓 이면 참, 참 이면 거짓
- 0 이외는 모두 참(1)

A	B	A && B	A B	!A
0 (거짓)	0 (거짓)	0 (거짓)	1 (참)	1 (참)
0 (거짓)	1 (참)	0 (거짓)	1 (참)	1 (참)
1 (참)	0 (거짓)	0 (거짓)	1 (참)	0 (거짓)
1 (참)	1 (참)	1 (참)	0 (거짓)	0 (거짓)

```
#include <stdio.h>

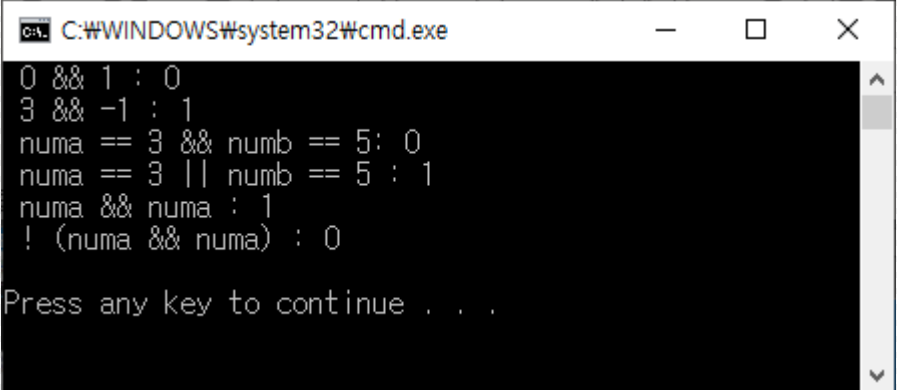
int main() {
    int numa = 10, numb = 5;

    printf(" 0 && 1 : %d \n", 0 && 1);
    printf(" 3 && -1 : %d \n", 3 && -1);

    printf(" numa == 3 && numb == 5: %d \n", numa == 3 && numb == 5);
    printf(" numa == 3 || numb == 5 : %d \n", numa == 3 || numb == 5);

    printf(" numa && numa : %d \n", numa && numa);
    printf(" ! (numa && numa) : %d \n", ! (numa && numa));

    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
0 && 1 : 0
3 && -1 : 1
numa == 3 && numb == 5: 0
numa == 3 || numb == 5 : 1
numa && numa : 1
! (numa && numa) : 0

Press any key to continue . . .
```

7. 할당 연산자

할당 연산자

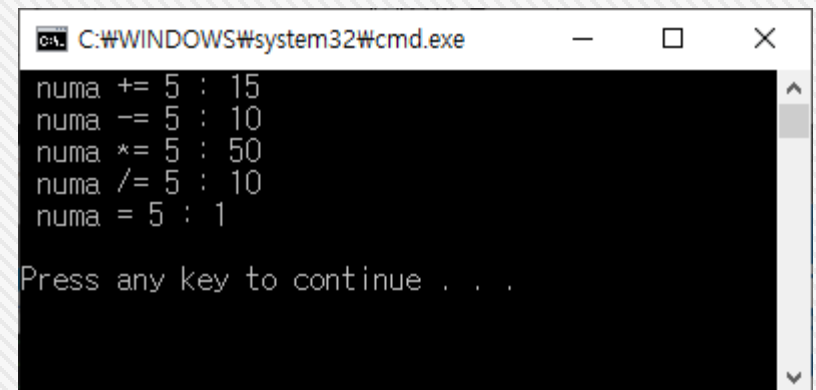
- 대입 연산자와 산술 연산자를 줄여서 사용

연산자	기능
+=	자신에 오른쪽 값을 더해 넣는다.
-=	자신에 오른쪽 값을 빼 넣는다.
*=	자신을 오른쪽 값으로 곱해 넣는다.
/=	자신을 오른쪽 값으로 나누어 몫을 넣는다.
%=	자신을 오른쪽 값으로 나누어 나머지를 넣는다.

```
#include <stdio.h>

int main() {
    int numa = 10, numb = 5;

    printf(" numa += 5 : %d \n", numa += 5);
    printf(" numa -= 5 : %d \n", numa -= 5);
    printf(" numa *= 5 : %d \n", numa *= 5);
    printf(" numa /= 5 : %d \n", numa /= 5);
    printf(" numa %= 5 : %d \n", numa %= 3);
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
numa += 5 : 15
numa -= 5 : 10
numa *= 5 : 50
numa /= 5 : 10
numa %= 5 : 1
Press any key to continue . . .
```

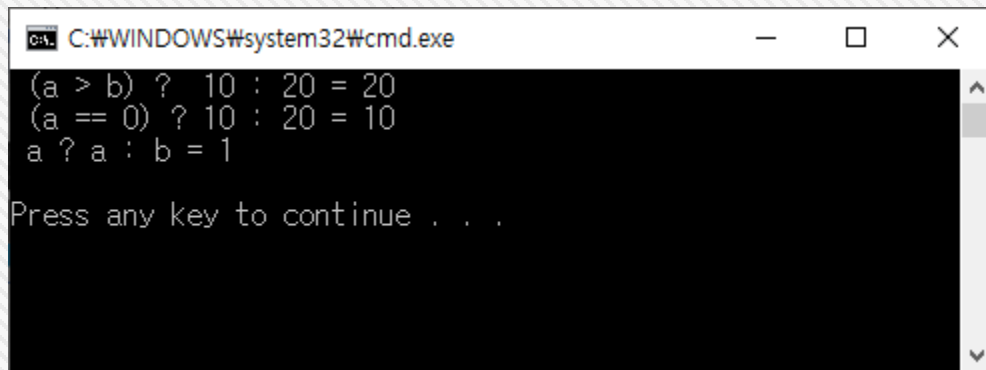

삼항 연산자

- 조건 ? 참일때의 값 : 거짓일때의 값;

```
#include <stdio.h>

int main() {
    int a = 0, b = 1, c = 0;

    printf(" (a > b) ? 10 : 20 = %d \n", (a > b) ? 10 : 20);
    printf(" (a == 0) ? 10 : 20 = %d \n", (a == 0) ? 10 : 20);
    printf(" a ? a : b = %d \n", a ? a : b);
    return 0;
}
```



C:\WINDOWS\system32\cmd.exe

```
(a > b) ? 10 : 20 = 20
(a == 0) ? 10 : 20 = 10
a ? a : b = 1
Press any key to continue . . .
```

수식 구성

- 논리 연산자의 특성을 이용한 조건 구성

data > 5

&&

data ++

data 가 3이면 data > 5 조건이 만족 하지 않으므로 data 값은 변경 없음
data 가 6이면 data > 5 조건이 만족 하여 data 값 증가 함.

```
#include <stdio.h>

int main() {
    int data = 3;

    printf( "data > 5 && data++ = %d\n", data > 5 && data++);
    printf( "data = %d\n", data);
    return 0;
}
```

data > 5 && data++ = 0
data = 0

```
#include <stdio.h>

int main() {
    int data = 7;

    printf( "data > 5 && data++ = %d\n", data > 5 && data++);
    printf( "data = %d\n", data);
    return 0;
}
```

data > 5 && data++ = 1
data = 8

data > 5

||

data ++

data > 5 조건이 만족 하지 않으므로 data 값은 증가함
data > 5 조건이 만족 하여 data 값은 변경 없음

```
#include <stdio.h>

int main() {
    int data = 3;

    printf( "data > 5 || data++ = %d\n", data > 5 || data++);
    printf( "data = %d\n", data);
    return 0;
}
```

data > 5 || data++ = 1
data = 4

```
#include <stdio.h>

int main() {
    int data = 3;

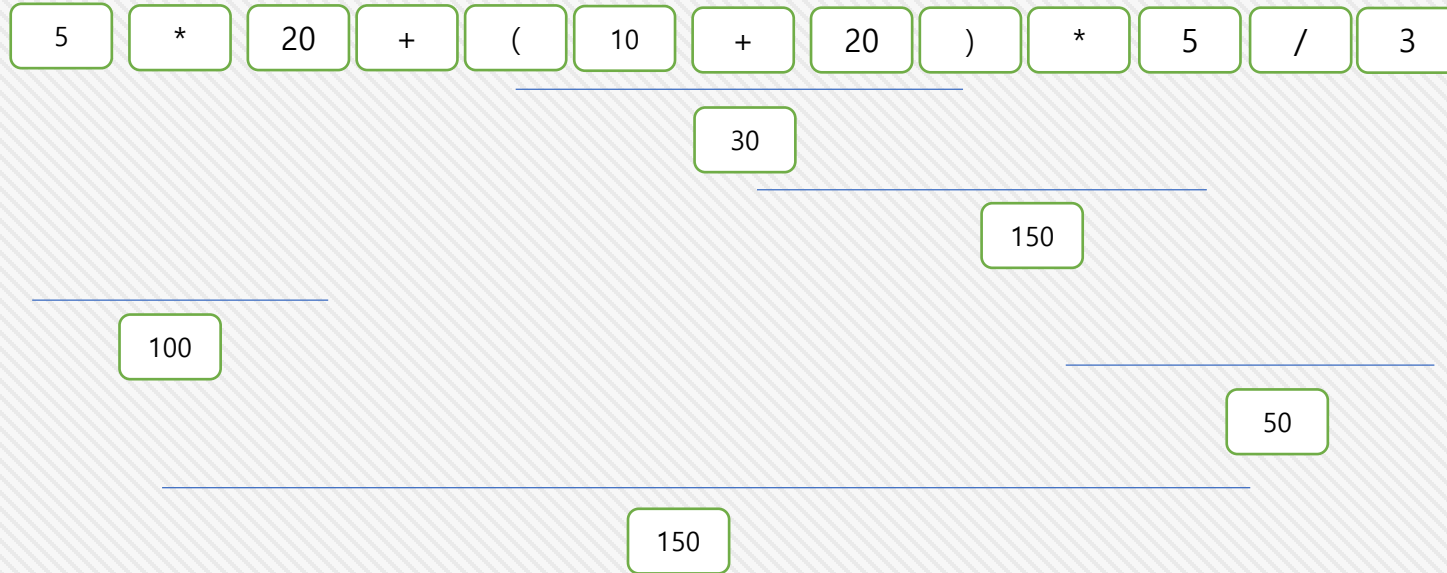
    printf( "data > 5 || data++ = %d\n", data > 5 || data++);
    printf( "data = %d\n", data);
    return 0;
}
```

data > 5 || data++ = 1
data = 7

9. 연산자 우선 순위

연산자 우선 순위

- 하나의 수식에 연산자를 여러 개 사용 하는 경우 어떤 연산자를 우선 실행 하는지 결정



```
#include <stdio.h>

int main() {
    printf( "%d\n", 5*20+(10+20)*5/3);
    return 0;
}
```

```
C:\WINDOWS\system32\cmd.exe
150
Press any key to continue . . .
```

9. 연산자 우선 순위

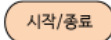

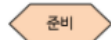
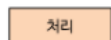
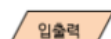

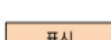
연산자 우선 순위

순위	종류	연산자	연산방향
1	괄호, 배열, 구조체	()(후위증가).[](후위감소)->	->
2	단항 연산자	(자료형) *(간접) &(주소) !(not) ~(비트) ++ -- +(부호) -(부호) sizeof	<-
3	승제 연산자	* / %	->
4	차감 연산자	+ -	->
5	시프트연산자	<< >>	->
6	비교 연산자	< <= > >=	->
7	등가 연산자	== !=	->
8	비트 연산자 AND	&	->
9	비트 연산자 XOR	^	->
10	비트 연산자 OR		->
11	논리 연산자 AND	&&	->
12	논리 연산자 OR		->
13	조건 연산자	? :	<-
14	대입 연산자	= *= /= += -= %= <<= >>= &= ^= =	<-
15	나열 연산자	,	->

- 우선 순위가 같은 연산자는 연산 방향 우선

순서도

- 순서도(flowchart)란 어떠한 일을 처리하는 과정을 순서대로 간단한 기호와 도형으로 도식화한 것을 의미
- 프로그래밍 전반에 걸쳐 분석, 기획, 디자인, 설계 단계에서 사용

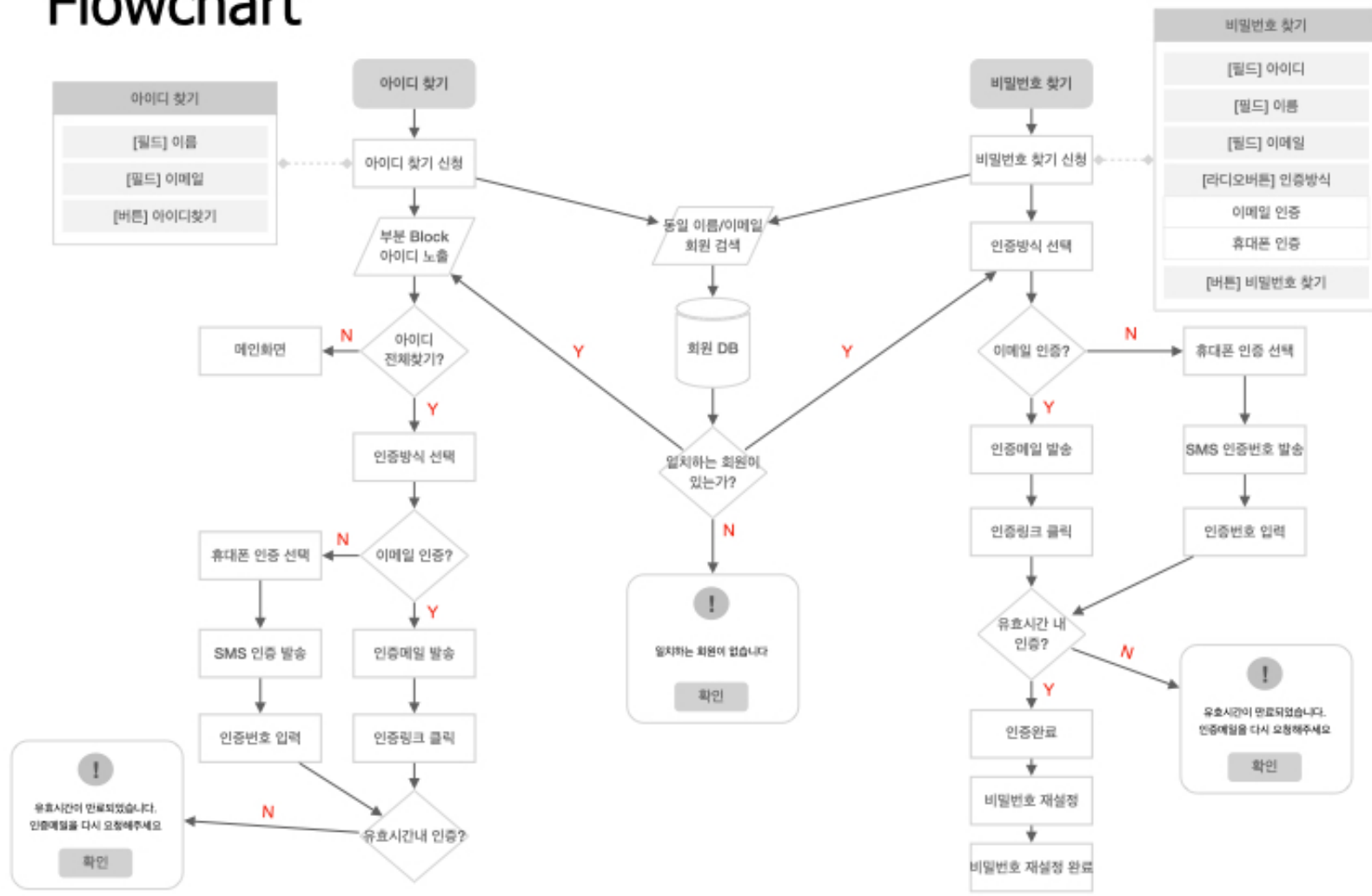
기호	명칭	설명
	단말	순서도의 시작과 끝을 나타냄.
	흐름선	순서도 기호 간의 연결 및 작업의 흐름을 표시함.
	준비	작업 단계 시작 전 해야 할 작업을 명시함.
	처리	처리해야 할 작업을 명시함.
	입출력	데이터의 입출력 시 사용함.
	의사 결정	비교 및 판단에 의한 논리적 분기를 나타냄.
	표시	화면으로 결과를 출력함.

기호	명칭	사용 용도	기호	명칭	사용 용도
	처리	각종 연산, 데이터 이동 등의 처리		터미널	순서도의 시작과 끝 표시
	연결자	흐름이 다른 곳과 연결되는 입출구를 나타냄		천공카드	천공카드의 입출력
	입출력	데이터의 입력과 출력		서류	서류를 매체로 하는 입출력 표시
	흐름선	처리의 흐름과 기호를 연결하는 기능		수동입력	콘솔에 의한 입력
	준비	기억장소, 초기값 등 작업의 준비 과정 나타냄		반복	조건을 만족하면 반복
	미리 정의된 처리	미리 정의된 처리로 옮길 때 사용		디스플레이	결과를 모니터로 나타냄

1. 순서도

4. 프로그램 제어 4-2. 순서도

Flowchart



- 프로그램의 흐름을 제어 할 수 있는 제어문에는 조건문과 반복문이 있다.
- 조건문 : 특정 조건을 부여 하여 조건에 만족 하는 경우와 불 만족하는 경우 문장을 수행함
- 반복문 : 특정 조건을 부여 하여 조건에 따라서 어떤 명령문들을 반복해서 수행함

조건문

- If 조건문
- If ~ else ~ 조건문
- 중첩된 조건문
- switch 조건문

반복문

- for 반복문
- while 반복문
- 중첩 반복문
- break, continue 제어문

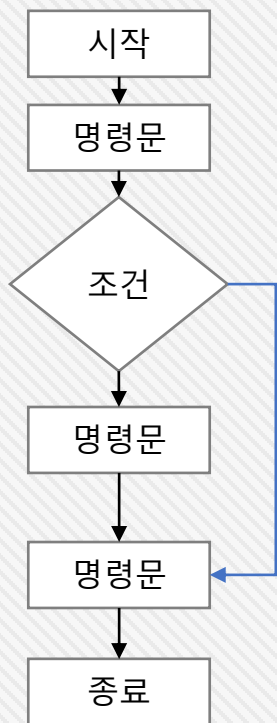
1. if 조건문

If 조건문

- 특정 조건이 만족 할 때 실행하는 문법.

If 조건문 구조

- if (조건 수식) 명령문;
- if (조건 수식) { 명령문들 };



```
int main()
int a = 0;
if ( a > 1 )
a = a + 1
printf(a)
return 0
```

```
#include <stdio.h>

int main() {
    int a = 0;
    if ( a > 1 ) a = a + 1;
    printf("a = %d", a);
    return 0;
}
```

C:\WINDOWS\system32\cmd.exe

```
a = 0
Press any key to continue . . .
```

```
#include <stdio.h>

int main() {
    int a = 2;
    if ( a > 1 ) {
        a = a + 1;
    };
    printf("a = %d", a);
    return 0;
}
```

C:\WINDOWS\system32\cmd.exe

```
a = 3
Press any key to continue . . .
```


1. if 조건문

if 조건문 주의 사항

- 세미콜론 ;의 위치에 따라서 결과는 틀려 짐

```
#include <stdio.h>

int main() {

    int a = 1;
    if ( a > 1 ) ;
        a ++;
    printf("a = %d", a);

    return 0;
}
```

a = 2

```
#include <stdio.h>

int main() {

    int a = 1;
    if ( a > 1 )
        a ++;
    printf("a = %d", a);

    return 0;
}
```

a = 3

- 대입 연산자와 관계 연산자

```
#include <stdio.h>

int main() {

    int a = 1;
    if ( a = 2 ) {
        a ++;
    }
    printf("a = %d", a);

    return 0;
}
```

a = 3

```
#include <stdio.h>

int main() {

    int a = 1;
    if ( a = 2 ) {
        a ++;
    }
    printf("a = %d", a);

    return 0;
}
```

a = 1

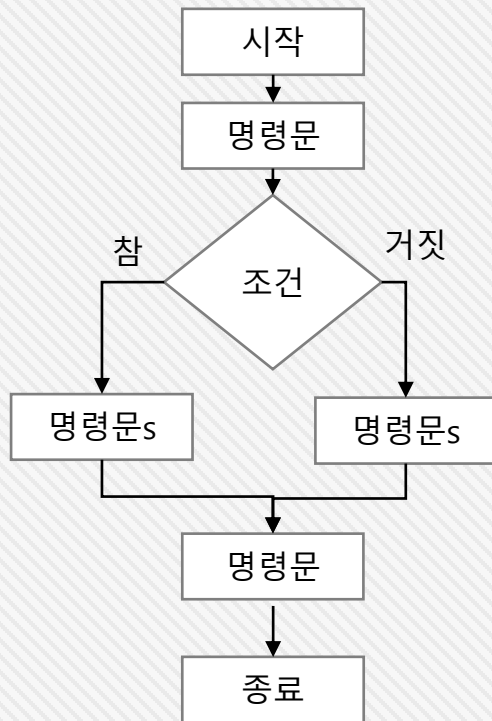
2. if ~ else 조건문

If ~ else 조건문

- 특정 조건이 만족 할 때 명령문 만족 하기 않을 때 명령문 실행

If 조건문 구조

- if (조건 수식) 명령문 else 명령문;
- if (조건 수식) { 명령문들 } else { 명령문들 } ;



```
#include <stdio.h>

int main() {

    int a = 4;
    if ( a < 5 ) a++;
    else a --;
    printf("a = %d", a);

    return 0;
}
```

```
C:\WINDOWS\system32\cmd.exe
a = 5
Press any key to continue . . .
```

```
#include <stdio.h>

int main() {

    int a = 5;
    if ( a < 5 ) a++;
    else a --;
    printf("a = %d", a);

    return 0;
}
```

```
C:\WINDOWS\system32\cmd.exe
a = 4
Press any key to continue . . .
```

```
#include <stdio.h>

int main() {

    int a = 5;
    if ( a < 5 ) {
        a++;
    } else {
        a--;
    }
    printf("a = %d", a);

    return 0;
}
```

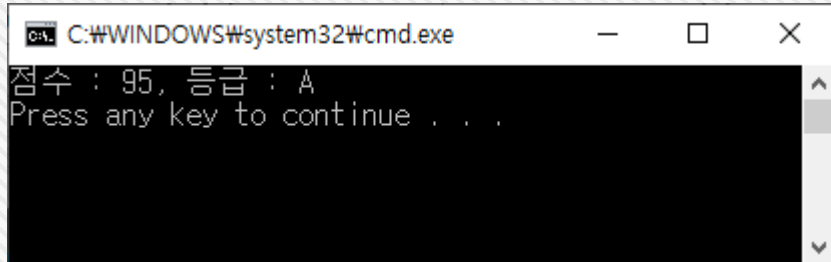
2. if ~ else 조건문

중복 구문은 한번에

```
#include <stdio.h>

int main() {
    int score = 95;
    char grade;
    if ( score >= 90 ) {
        grade = 'A';
        printf("점수 : %d, 등급 : %c", score, grade);
    } else {
        grade = 'B';
        printf("점수 : %d, 등급 : %c", score, grade);
    }

    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
점수 : 95, 등급 : A
Press any key to continue . . .
```

```
#include <stdio.h>

int main() {
    int score = 95;
    char grade;
    if ( score >= 90 ) {
        grade = 'A';
    } else {
        grade = 'B';
    }
    printf("점수 : %d, 등급 : %c", score, grade);
    return 0;
}
```

삼항 연산자로 표시

```
#include <stdio.h>

int main() {
    int score = 95;
    char grade;
    grade = ( score >= 90 ) ? 'A' : 'B';
    printf("점수 : %d, 등급 : %c", score, grade);
    return 0;
}
```

2. if ~ else 조건문

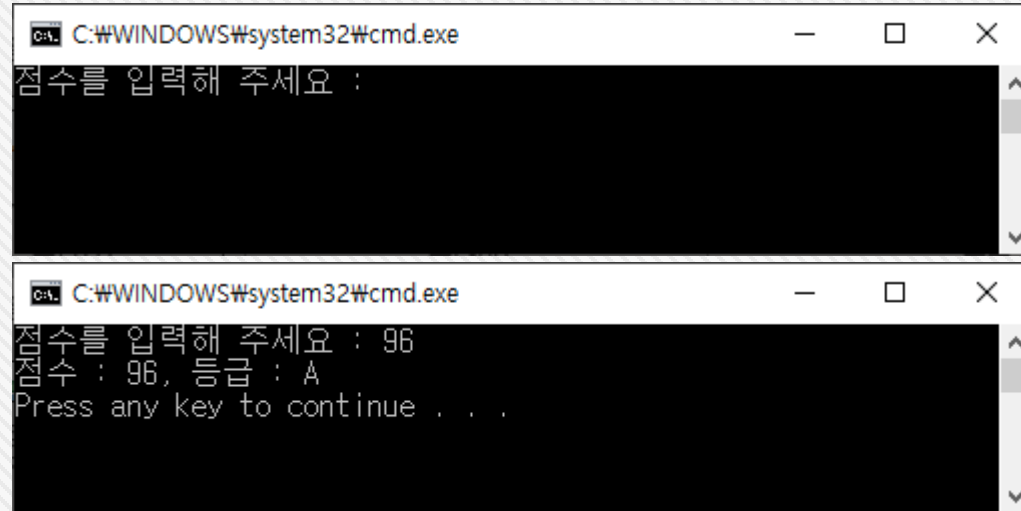
Keyboard로 점수를 입력 하여 등급 출력

```
#include <stdio.h>

int main() {
    int score;
    char grade;

    printf("점수를 입력해 주세요 : ");
    scanf("%d", &score);

    grade = ( score >= 90 ) ? 'A' : 'B';
    printf("점수 : %d, 등급 : %c", score, grade);
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
점수를 입력해 주세요 :

C:\WINDOWS\system32\cmd.exe
점수를 입력해 주세요 : 96
점수 : 96, 등급 : A
Press any key to continue . . .
```

2. if ~ else 조건문

4. 프로그램 제어 4-3. 조건문

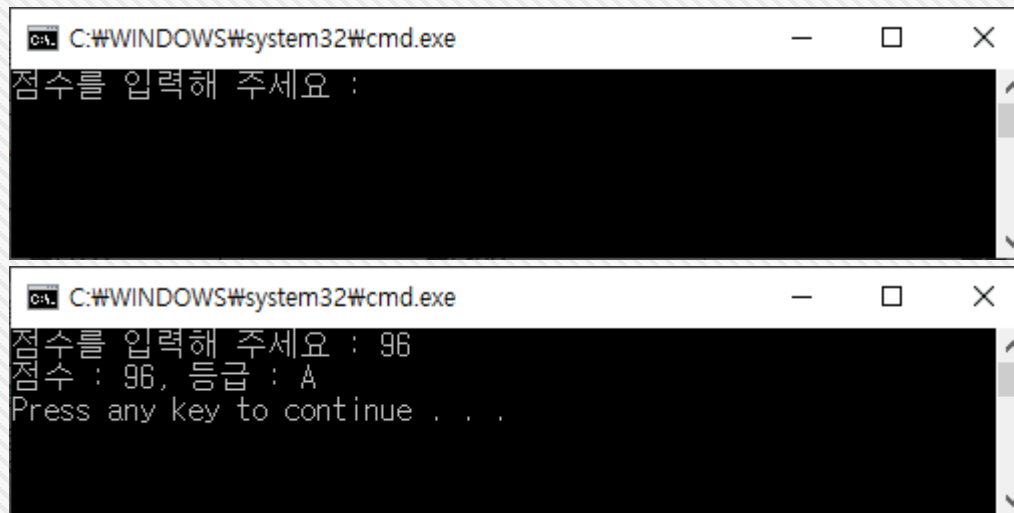
Keyboard로 점수를 입력 하여 등급 출력

```
#include <stdio.h>

int main() {
    int score;
    char grade;

    printf("점수를 입력해 주세요 : ");
    scanf("%d", &score);

    grade = ( score >= 90 ) ? 'A' : 'B';
    printf("점수 : %d, 등급 : %c", score, grade);
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
점수를 입력해 주세요 :

점수를 입력해 주세요 : 96
점수 : 96, 등급 : A
Press any key to continue . . .
```

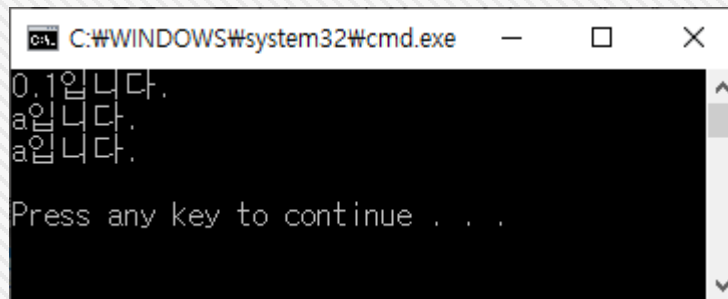
실수, 문자 비교

```
float num1 = 0.1f;
char c1 = 'a';

if (num1 == 0.1f) // 실수 비교
    printf("0.1입니다.\n");

if (c1 == 'a') // 문자 비교
    printf("a입니다.\n");

if (c1 == 97) // 문자를 ASCII 코드로 비교
    printf("a입니다.\n");
```



```
C:\WINDOWS\system32\cmd.exe
0.1입니다.
a입니다.
a입니다.
Press any key to continue . . .
```

3. 중첩된 조건문

4. 프로그램 제어

4-3. 조건문

중첩된 조건문

```
#include <stdio.h>

int main() {
    int score = 75;
    char grade;
    if ( score >= 90 ) {
        grade = 'A';
    } else {
        if (score >= 80) {
            grade = 'B';
        } else {
            if (score >= 70) {
                grade = 'C';
            } else {
                if (score >= 60) {
                    grade = 'D';
                } else {
                    grade = 'F';
                }
            }
        }
    }

    printf("점수 : %d, 등급 : %c", score, grade);
    return 0;
}
```

```
#include <stdio.h>

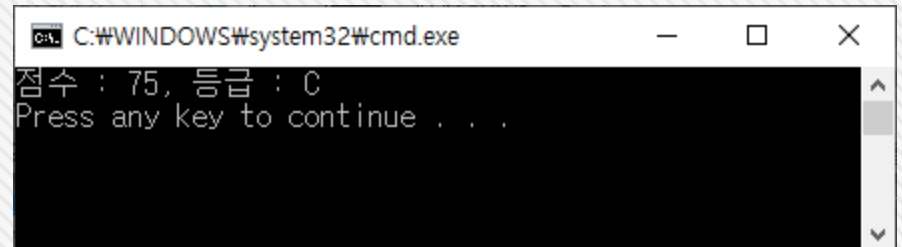
int main() {
    int score = 75;
    char grade;
    if ( score >= 90 ) grade = 'A';
    else
        if (score >= 80) grade = 'B';
        else {
            if (score >= 70) grade = 'C';
            else {
                if (score >= 60) grade = 'D';
                else grade = 'F';
            }
        }

    printf("점수 : %d, 등급 : %c", score, grade);
    return 0;
}
```

```
#include <stdio.h>

int main() {
    int score = 75;
    char grade;
    if ( score >= 90 ) grade = 'A';
    else if (score >= 80) grade = 'B';
    else if (score >= 70) grade = 'C';
    else if (score >= 60) grade = 'D';
    else grade = 'F';

    printf("점수 : %d, 등급 : %c", score, grade);
    return 0;
}
```



A screenshot of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window shows the output of the program: "점수 : 75, 등급 : C" followed by "Press any key to continue . . .". The cursor is positioned at the end of the second line.

4. switch

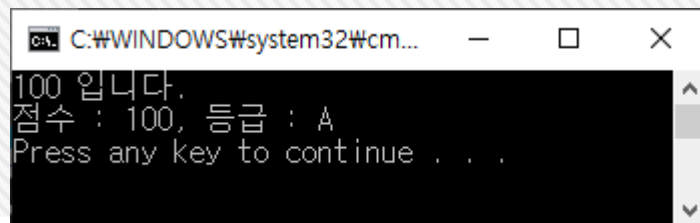
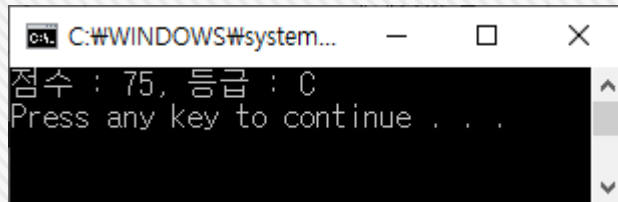
4. 프로그램 제어

4-3. 조건문

switch

- if문의 비효율적인 문제를 해결 하기 위함
- 변수 값이 이미 정해져 있는 상수와 비교 할 때 사용
- break 를 만나면 switch 블록을 벗어남

```
명령문;
switch(수식 또는 변수) {
    case 상수1:
        명령문;
        break;
    case 상수2:
        명령문;
        break;
    default:
        명령문;
        break;
}
명령문;
```



```
#include <stdio.h>

int main() {
    int score = 75;
    char grade;

    switch(score/10) {
        case 10:
        case 9:
            grade = 'A';
            break;
        case 8:
            grade = 'B';
            break;
        case 7:
            grade = 'C';
            break;
        case 6:
            grade = 'D';
            break;
        default:
            grade = 'A';
    }

    printf("점수 : %d, 등급 : %c", score, grade);
    return 0;
}
```

```
#include <stdio.h>

int main() {
    int score = 100;
    char grade;

    switch(score/10) {
        case 10:
            printf("100 입니다.\n");
        case 9:
            grade = 'A';
            break;
        case 8:
            grade = 'B';
            break;
        case 7:
            grade = 'C';
            break;
        case 6:
            grade = 'D';
            break;
        default:
            grade = 'A';
    }

    printf("점수 : %d, 등급 : %c", score, grade);
    return 0;
}
```

반복문

- “1에서 5까지 더하기”에서 1씩 더하는 작업이 반복 작업이 존재 하는 경우 사용 하는 문법
- 시작(1에서) , 종결(5까지), 반복 (1씩 증가), 더하기(명령문) 로 구성됨

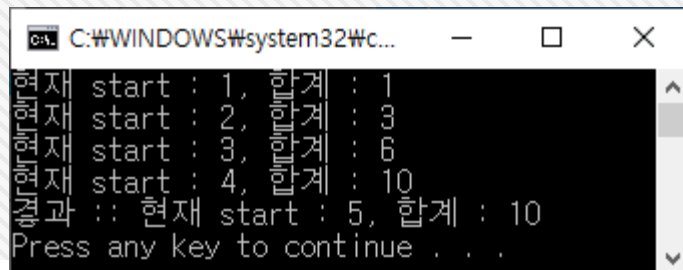
for

- for(시작 조건; 종결조건; 조건변화 수식)

```
#include <stdio.h>

int main() {
    int start, sum = 0;
    for (start = 1; start < 5; start++) {
        sum = sum + start;
        printf("현재 start : %d, 합계 : %d\n", start, sum);
    }

    printf("결과 :: 현재 start : %d, 합계 : %d", start, sum);
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
현재 start : 1, 합계 : 1
현재 start : 2, 합계 : 3
현재 start : 3, 합계 : 6
현재 start : 4, 합계 : 10
결과 :: 현재 start : 5, 합계 : 10
Press any key to continue . . .
```

- 시작 조건이 선언 시점에 할당 되어 있으면 생략 가능

```
#include <stdio.h>

int main() {
    int start = 1, sum = 0;
    for (; start < 5; start++) {
        sum = sum + start;
        printf("현재 start : %d, 합계 : %d\n", start, sum);
    }

    printf("결과 :: 현재 start : %d, 합계 : %d", start, sum);
    return 0;
}
```


for

- for문 안에 사용 하는 변수가 선언만 되어 있으며 시작조건에 할당 가능함
- for문을 무한 루프 형식과 break를 이용

```
#include <stdio.h>

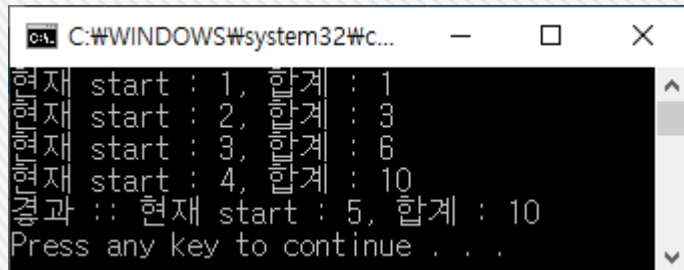
int main() {
    int start, sum;
    for (start = 1, sum = 0; start < 5; start++) {
        sum = sum + start;
        printf("현재 start : %d, 합계 : %d\n", start, sum);
    }

    printf("결과 :: 현재 start : %d, 합계 : %d", start, sum);
    return 0;
}
```

```
#include <stdio.h>

int main() {
    int start = 1, sum = 0;
    for (;;) {
        sum = sum + start;
        printf("현재 start : %d, 합계 : %d\n", start, sum);
        start++;
        if (start > 4) break;
    }

    printf("결과 :: 현재 start : %d, 합계 : %d", start, sum);
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
현재 start : 1, 합계 : 1
현재 start : 2, 합계 : 3
현재 start : 3, 합계 : 6
현재 start : 4, 합계 : 10
결과 :: 현재 start : 5, 합계 : 10
Press any key to continue . . .
```

2. while

while

- 종결 조건만으로도 반복문 실행
- while(종결 조건) {
 명령문 s
}

```
#include <stdio.h>

int main() {
    int start = 1, sum = 0 ;
    while (start < 5) {
        sum = sum + start;
        printf("현재 start : %d, 합계 : %d\\n", start, sum);
        start ++;
    }

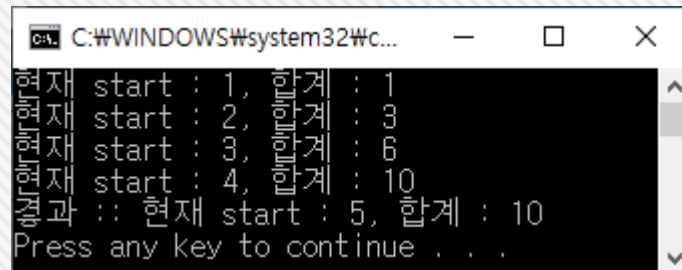
    printf("결과 :: 현재 start : %d, 합계 : %d", start, sum);
    return 0;
}
```

- while문을 무한 루프 형식과 break를 이용

```
#include <stdio.h>

int main() {
    int start = 1, sum = 0 ;
    while (1) {
        sum = sum + start;
        printf("현재 start : %d, 합계 : %d\\n", start, sum);
        start ++;
        if (start >= 5) break;
    }

    printf("결과 :: 현재 start : %d, 합계 : %d", start, sum);
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
현재 start : 1, 합계 : 1
현재 start : 2, 합계 : 3
현재 start : 3, 합계 : 6
현재 start : 4, 합계 : 10
결과 :: 현재 start : 5, 합계 : 10
Press any key to continue . . .
```

3. do ~ while

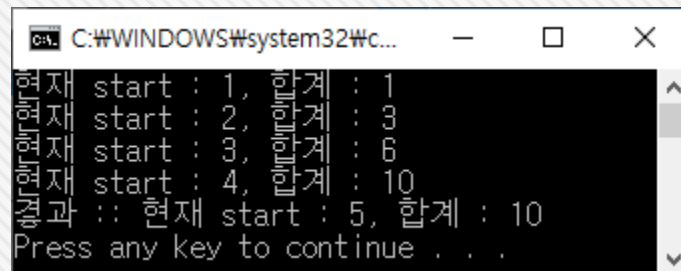
do ~ while

- 종결 조건만으로도 반복문 실행하는 것은 while문과 동일 하지만 무조건 한번 실행
- do {
 명령문 s
} while(종결 조건)

```
#include <stdio.h>

int main() {
    int start = 1, sum = 0 ;
    do {
        sum = sum + start;
        printf("현재 start : %d, 합계 : %d\n", start, sum);
        start ++;
    } while ( start < 5 );

    printf("결과 :: 현재 start : %d, 합계 : %d", start, sum);
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
현재 start : 1, 합계 : 1
현재 start : 2, 합계 : 3
현재 start : 3, 합계 : 6
현재 start : 4, 합계 : 10
결과 :: 현재 start : 5, 합계 : 10
Press any key to continue . . .
```

4. break, continue

4. 프로그램 제어

4-4. 반복문

break

- 반복문을 벗어나기 위함
- 하나의 블록만 벗어남
- 구구단 출력

```
#include <stdio.h>
int main() {
    int start , second ;
    for ( start = 2 ; start < 10; start++) {
        printf("%d 구단\n", start );
        for ( second = 1 ; second < 10; second++) {
            printf("%d * %d = %d\n", start, second, start * second);
        }
    }
    return 0;
}
```

- 구구단 출력 – 각각의 구구단 2단 3 까지만 출력

- 구구단 출력 – 각각의 구구단 3 까지만 출력

```
#include <stdio.h>

int main() {
    int start , second ;
    for ( start = 2 ; start < 10; start++) {
        printf("%d 구단\n", start );
        for ( second = 1 ; second < 10; second++) {
            printf("%d * %d = %d\n", start, second, start * second);
            if ( second > 2 ) break;
        }
    }
    return 0;
}
```

```
#include <stdio.h>

int main() {
    int start , second ;
    for ( start = 2 ; start < 10; start++) {
        printf("%d 구단\n", start );
        for ( second = 1 ; second < 10; second++) {
            printf("%d * %d = %d\n", start, second, start * second);
            if ( second > 2 ) break;
        }
        if ( start <= 2 ) break;
    }
    return 0;
}
```

4. break, continue

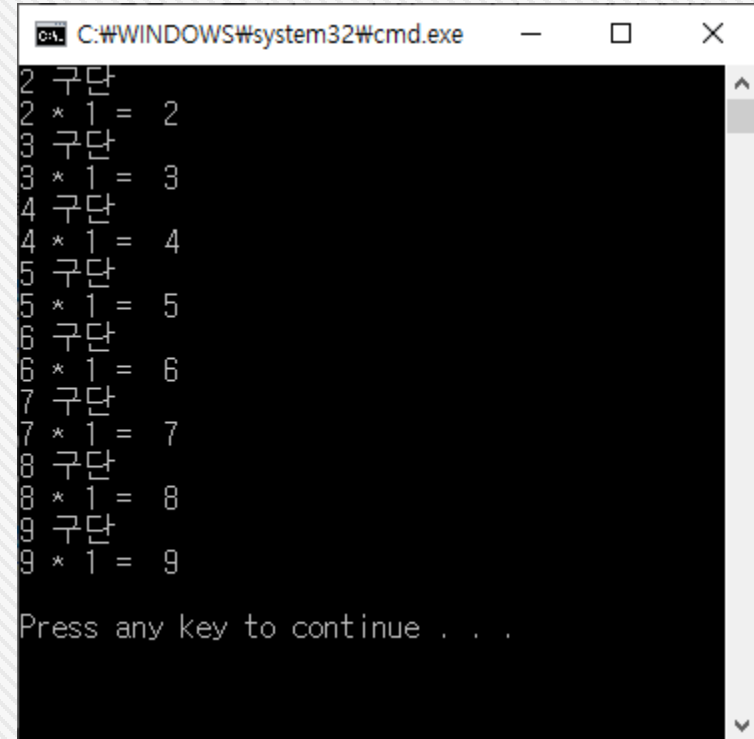
continue

- 1회성 최소
 - 구구단 출력 - 각각의 구구단 1만 출력

```
#include <stdio.h>

int main() {
    int start, second;
    for (start = 2; start < 10; start++) {
        printf("%d 구단\n", start);
        for (second = 1; second < 10; second++) {
            if (second > 1) continue;
            printf("%d * %d = %d\n", start, second, start * second);
        }
    }

    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
2 구단
2 * 1 = 2
3 구단
3 * 1 = 3
4 구단
4 * 1 = 4
5 구단
5 * 1 = 5
6 구단
6 * 1 = 6
7 구단
7 * 1 = 7
8 구단
8 * 1 = 8
9 구단
9 * 1 = 9

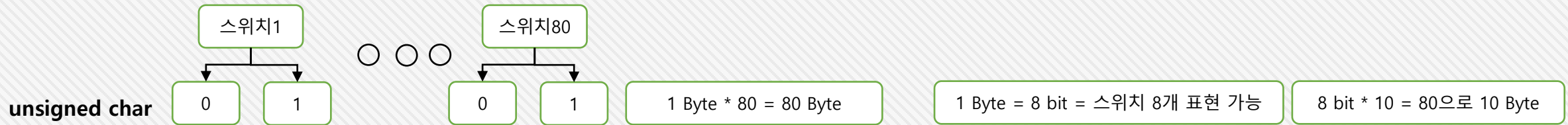
Press any key to continue . . .
```

1. 비트 연산

4. 프로그램 제어 4-5. 비트연산

비트 연산이 필요한 이유

- 메모리 절약.
- 컴퓨터는 0 또는 1로 저장 하고 처리를 하는데 C언어의 최소 저장 공간은 1Byte (8 bit) 임.
- 만약 전원의 on/off를 표시 하기 위해서는 1byte가 필요 한데 전원 스위치가 80개 이면 80byte가 필요 한데 이것을 bit로 처리 하게 하며 10byte로 처리 가능 하므로 메모리를 절약 할 수 있음



- C언어는 2진수 상수를 제공하는 방법이 없으므로 16진수를 사용 하여 프로그램 함.

2 진수	16 진수	2 진수	16 진수	2 진수	16 진수	2 진수	16 진수
0000	0	0100	4	1000	8	1100	C (12)
0001	1	0101	5	1001	9	1101	D (13)
0010	2	0110	6	1010	A (10)	1110	E (14)
0011	3	0111	7	1011	B (11)	1111	F (15)

unsigned char
= 1 Byte

0 (7 bit)	1 (6 bit)	0 (5 bit)	1 (4 bit)	1 (3 bit)	0 (2 bit)	1 (0 bit)	1 (0 bit)
5				B			

C 언어 표현 : unsigned char a = 0x5B ; => 십진수 : $5 * (16^1) + 11 * 16^0 = 91$

1. 시프트 연산자

시프트 연산자

- << (오른쪽 에서 왼쪽) , >> (왼쪽에서 오른쪽)을 사용 하여서 지정한 비트 수 만큼 이동

data = data << 2

부호 없는 값

0	0	1	0	0	0	1	0
---	---	---	---	---	---	---	---

0x22

0	0	1	0	0	0	1	0		
---	---	---	---	---	---	---	---	--	--

overflow data 버려 짐

1	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

0x88

$$8 * 16^1 + 8 * 16^0 = 136$$

반공간 0으로 채워 짐

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

```
void main() {  
    unsigned char data = 0x22;  
    printf("십진수 : %d, 16진수 : %#x \n", data, data);  
    data = data << 2;  
    printf("십진수 : %d, 16진수 : %#x \n", data, data);  
}
```

십진수 : 34, 16진수 : 0x22
십진수 : 136, 16진수 : 0x88

부호 있는 값

0	0	1	0	0	0	1	0
---	---	---	---	---	---	---	---

0x22

0	1	0	0	0	1	0	0
---	---	---	---	---	---	---	---

1	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

0x88

$$8 * 16^1 - 8 * 16^0 = 120$$

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

```
void main() {  
    char data = 0x22;  
    printf("십진수 : %d, 16진수 : %#x \n", data, data);  
    data = data << 2;  
    printf("십진수 : %d, 16진수 : %#x \n", data, data);  
}
```

십진수 : 34, 16진수 : 0x22
십진수 : -120, 16진수 : 0xffffffff88

- 음수로 변화는 경우 가 발생 => 예측 할 수 없는 값 발생 => 사용 하지 말 아함

2. 곱셈/나눗셈

4. 프로그램 제어 4-6. 시프트연산자

시프트 연산자

- << (오른쪽 에서 왼쪽)

0	0	0	0	0	0	0	1	0x01	1	* 2 ⁿ
0	0	0	0	0	0	1	0	0x02	2	
0	0	0	0	0	1	0	0	0x04	4	
0	0	0	0	1	0	0	0	0x08	8	
0	0	0	1	0	0	0	0	0x10	16	
0	0	1	0	0	0	0	0	0x20	32	
0	1	0	0	0	0	0	0	0x40	64	
1	0	0	0	0	0	0	0	0x80	128	

- >> 왼쪽에서 오른쪽)

1	0	0	0	0	0	0	0	0x80	128	/ 2 ⁿ
0	1	0	0	0	0	0	0	0x40	64	
0	0	1	0	0	0	0	0	0x20	32	
0	0	0	1	0	0	0	0	0x10	16	
0	0	0	0	1	0	0	0	0x08	8	
0	0	0	0	0	1	0	0	0x04	4	
0	0	0	0	0	0	1	0	0x02	2	
0	0	0	0	0	0	0	1	0x01	1	
0	0	0	0	0	0	0	0	0x00	0	

```
#include <stdio.h>

void main() {
    unsigned char data02 = 4; // 0x04

    unsigned char data02Result = data02 << 2;
    printf("십진수 : %d, data02 << 2 : %d \n", data02, data02Result);
}
```

십진수 : 4, data02 << 2 : 16

```
#include <stdio.h>

void main() {
    unsigned char data03 = 128; // 0x80;

    unsigned char data03Result = data03 >> 2;
    printf("십진수 : %d, data03 >> 2 : %d \n", data03, data03Result);
}
```

십진수 : 128, data03 >> 2 : 32

1. Bit 연산자

bit 연산자

- bit 단위로 연산을 할 때 사용
- 논리 연산자와 구별 하기 위해서 AND 연산 `&`, OR 연산 `|`, NOT 연산 `~`, XOR 연산 `^` 로 표시

연산자	설명
<code>&</code>	비트 AND
<code> </code>	비트 OR
<code>^</code>	비트 XOR (배타적 OR, Exclusive OR)
<code>~</code>	비트 NOT

A	B	AND(&)	OR()	XOR(^)	NOT (~)
0	0	0	1	0	1
0	1	0	1	1	
1	0	0	1	1	0
1	1	1	0	0	

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

```
void main() {
```

```
    unsigned char a = 0x23;
```

```
    unsigned char b = 0x42;
```

```
    unsigned char c = a & b;
```

```
    unsigned char d = a | b;
```

```
    unsigned char e = a ^ b;
```

```
    unsigned char f = ~a;
```

```
    printf("a (%#x): %d, b (%#x) : %d  \n", a, a, b, b);
```

```
    printf("AND(a & b) = c : %d (%#x) \n", c,c);
```

```
    printf("OR(a & b) = c : %d (%#x) \n", d,d);
```

```
    printf("XOR(a & b) = c : %d (%#x) \n", e,e);
```

```
    printf("NOT(~a) = c : %d (%#x) \n", f,f);
```

```
}
```

```
a (0x23): 35, b (0x42) : 66
AND(a & b) = c : 2 (0x2)
OR(a & b) = c : 99 (0x63)
XOR(a & b) = c : 97 (0x61)
NOT(~a) = c : 220 (0xdc)
```

	10진수	16진수	값 (1 byte)	
a	35	0x23	0010	0011
b	66	0x42	0100	0010
AND	2	0x2	0000	0010
OR	99	0x63	0110	0011
XOR	97	0x61	0110	0001
NOT	220	0xdc	1101	1100

2. Bit 연산자 + 할당 연산자

bit 연산자

- 할당 연산자와 혼합 해서 사용 할 수 있음

&=	비트 AND 연산 후 할당
=	비트 OR 연산 후 할당
^=	비트 XOR 연산 후 할당
<<=	비트를 왼쪽으로 시프트한 후 할당
>>=	비트를 오른쪽으로 시프트한 후 할당

```
C:\WINDOWS\system32\cmd.exe
AND (a &= 5)   = a : 0      (0)
OR  (b |= 5)   = b : 71     (0)
XOR (c ^= 5)   = c : 71     (0x47)
<< (d <<= 5)   = d : 64     (0x40)
>> (e >>= 5)   = e : 2      (0x2)

Press any key to continue . . .
```

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

```
void main() {
```

```
    unsigned char a = 0x42;
```

```
    unsigned char b = 0x42;
```

```
    unsigned char c = 0x42;
```

```
    unsigned char d = 0x42;
```

```
    unsigned char e = 0x42;
```

```
    a &= 5;
```

```
    b |= 5;
```

```
    c ^= 5;
```

```
    d <<= 5;
```

```
    e >>= 5;
```

```
    printf("AND (a &= 5)   = a : %d \t(%#x) \n", a, a);
```

```
    printf("OR  (b |= 5)   = b : %d \t(%#x) \n", b, a);
```

```
    printf("XOR (c ^= 5)   = c : %d \t(%#x) \n", c, c);
```

```
    printf("<< (d <<= 5)   = d : %d \t(%#x) \n", d, d);
```

```
    printf(">> (e >>= 5)   = e : %d \t(%#x) \n", e, e);
```

```
}
```

3. Bit 연산자 활용

iot 예제

- iot 장비에서 8개의 장비에서 전원 on/off 신호가 전달 된다. On :1, Off: 0 전원이 켜져 있는 장비를 표시 하라.

8번 장비	7번 장비	6번 장비	5번 장비	4번 장비	3번 장비	2번 장비	1번 장비
0	1	0	0	1	1	1	0

```
#include <stdio.h>

void main() {
    unsigned char deviceValue ;

    printf("16 진수 문자를 입력 하세요 : ");
    scanf("%x", &deviceValue) ;
    printf("\n입력 값 : %#x \n", deviceValue);
    if (deviceValue & 1) {
        printf ( " 1 번 장비 on , %#x\n", deviceValue & 1);
    };

    if (deviceValue & 2) {
        printf ( " 2 번 장비 on , %d\n", deviceValue & 2);
    };

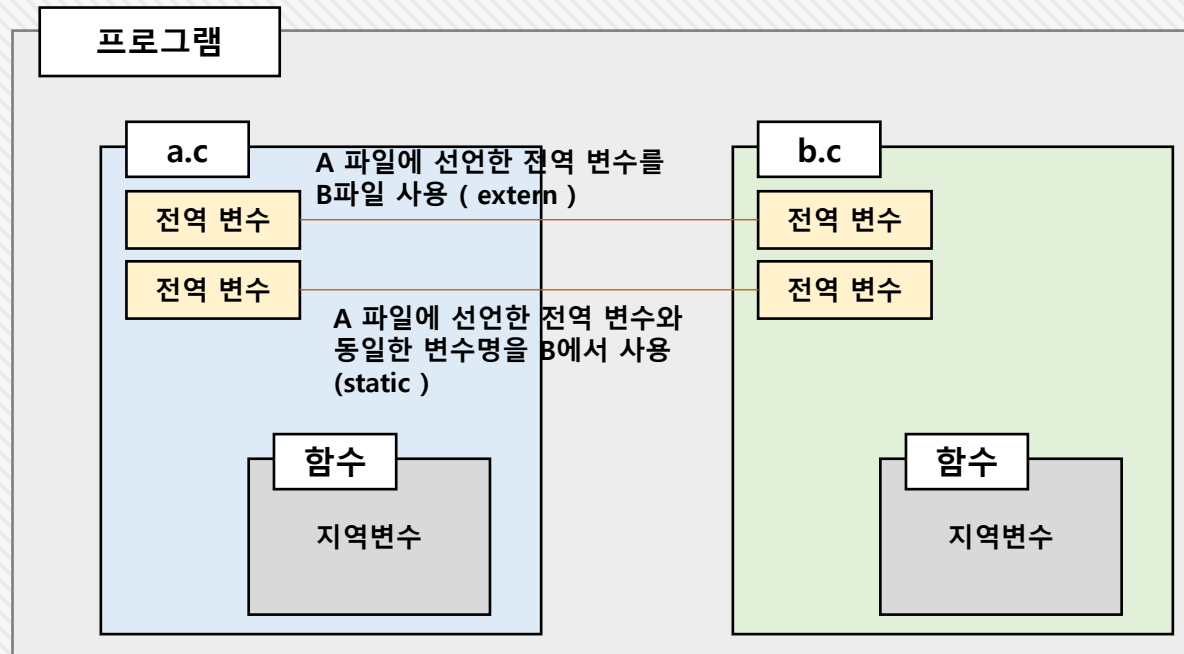
    if (deviceValue & 4) {
        printf ( " 3 번 장비 on , %d\n", deviceValue & 4);
    };

    .....
}
```

1. 변수 범위

변수 범위

- 변수의 선언 위치에 따라서 변수의 수명이 지역 범위를 가지는 지역 변수와 프로그램 전체에 영향을 주는 전역 변수.
- 변수 선언 시 `extern`, `static`, `const` keyword를 사용 하면 의미가 변경 됨



1. 변수 범위

4. 프로그램 제어 4-8. 변수 범위

지역 변수

- 함수 내부에 선언된 변수는 다른 외부 함수에서 사용 할 수 없다.
- 함수 내부에 선언된 변수도 함수 내부에 if 함수 내에 선언된 변수를 if 함수 외부에서 사용 할 수 없다.

```
1  #include <stdio.h>
2
3  int funNum(); // 함수 원형
4
5  void main() {
6      int mainVar ;
7      printf("함수 호출 결과 : %d", funNum());
8  }
9
10 int funNum() {
11     return mainVar + 5;
12 }
```

⊗ 125_localVariable.c 문제 3개 중 3개
식별자 "mainVar"이 (가) 정의되어 있지 않습니다. C/C++(20)

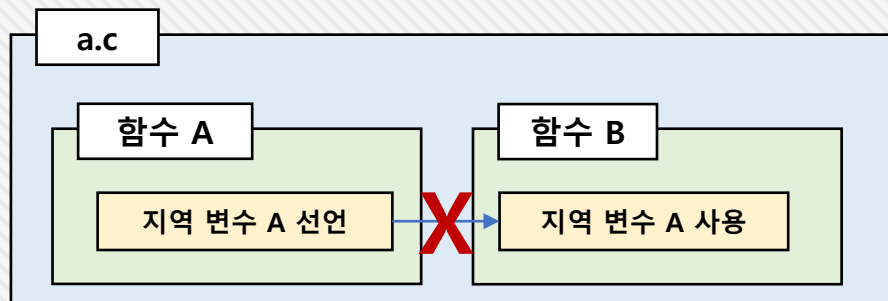
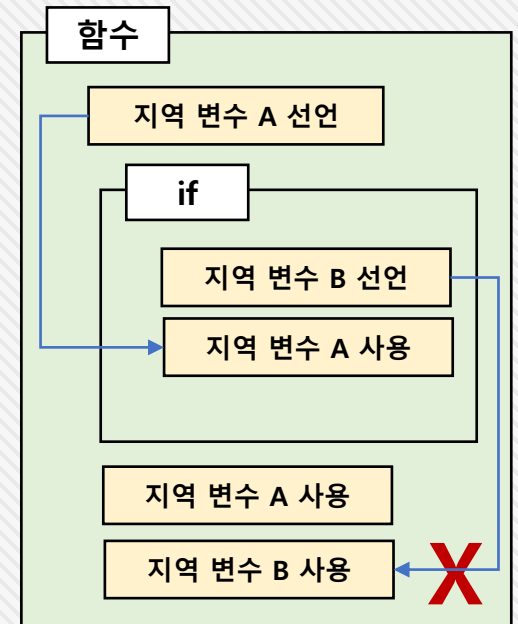
```
#include <stdio.h>
void main() {
    int mainVar = 0 ;

    if (1) {
        int ifVar = 1;
        mainVar += ifVar;
    }

    printf("if 블록 내부 선언 변수 : %d", ifVar);
}

25_localVariable.c 문제 1개 중 1개
자 "ifVar"이 (가) 정의되어 있지 않습니다. C/C++(20)

printf("if 블록 외부 선언 변수 : %d", mainVar);
}
```

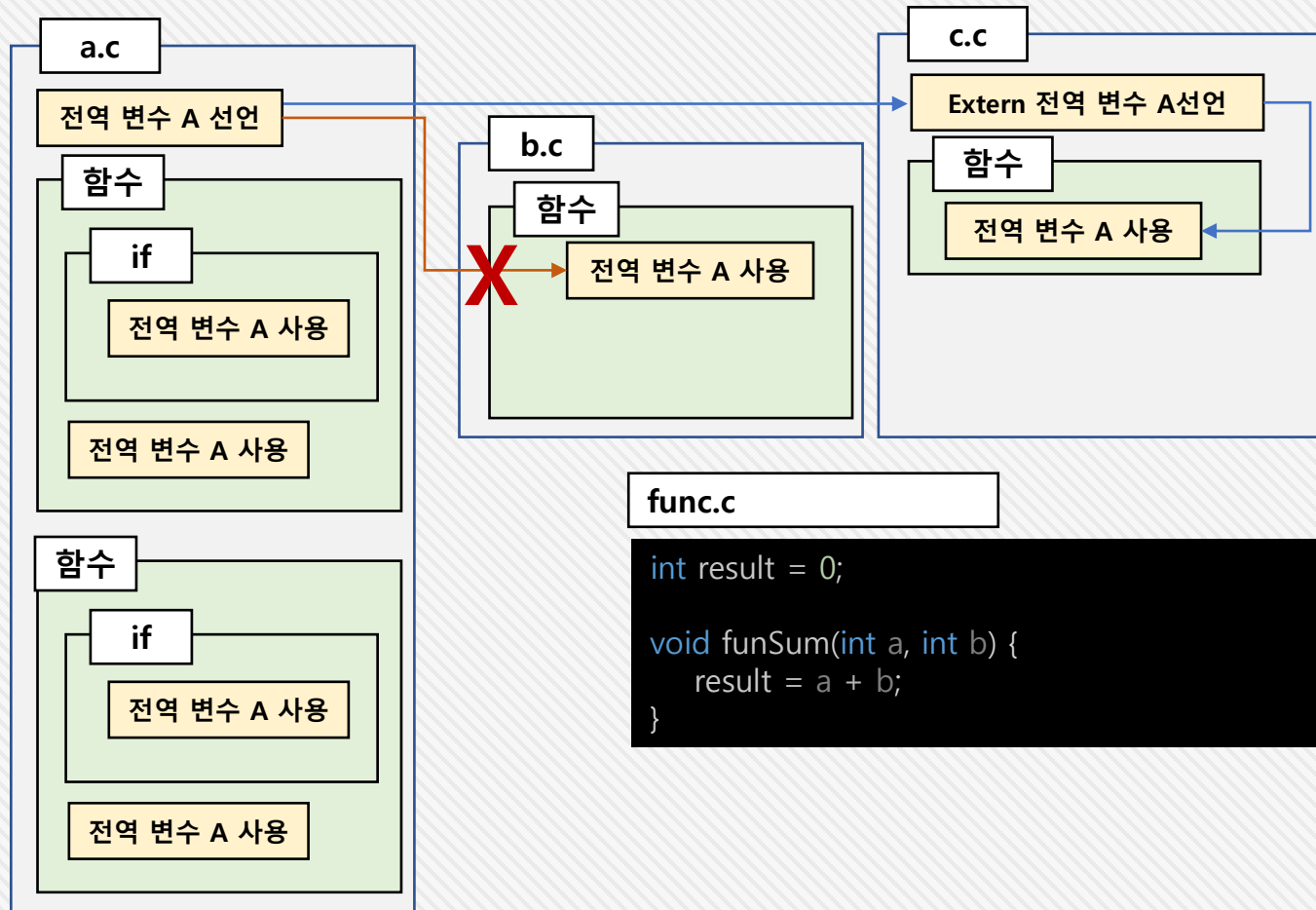


1. 변수 범위

4. 프로그램 제어 4-8. 변수 범위

전역 변수 - Project

- 하나의 파일 안에서 함수 밖에 선언된 변수
- 함수 밖에 선언된 변수는 해당 파일에 함수, 함수 내 블록 (if, for ...) 모두 사용 가능
- 변수가 선언된 파일이외의 파일에서는 할 수 없음 -> 사용 하기 위해서 extern 선언 필요



```
1  #include <stdio.h>
2
3  void funSum(a, b);
4
5  void main() {
6      funSum(1, 1);
7      printf("결과 : %d", result);
```

⊗ 125_globalVarA.c 문제 1개 중 1개

식별자 "result"이 (가) 정의되어 있지 않습니다. C/C++(20)

```
8  }
```

main.c

```
#include <stdio.h>

void funSum(a, b);
extern int result;

void main() {
    funSum(1, 1);
    printf("결과 : %d", result);
}
```

1. 변수 범위

4. 프로그램 제어 4-8. 변수 범위

전역 변수 - static

- 전역 변수를 특정 파일에 국한
- 함수 안에서 사용 하는 경우 해당 함수 내 에서만 전역화 (지역변수와 차이점 ?)

```
#include <stdio.h>

extern void funSum(int a, int b);
extern void fun_Printf();
extern int result;
void funSumAdd();

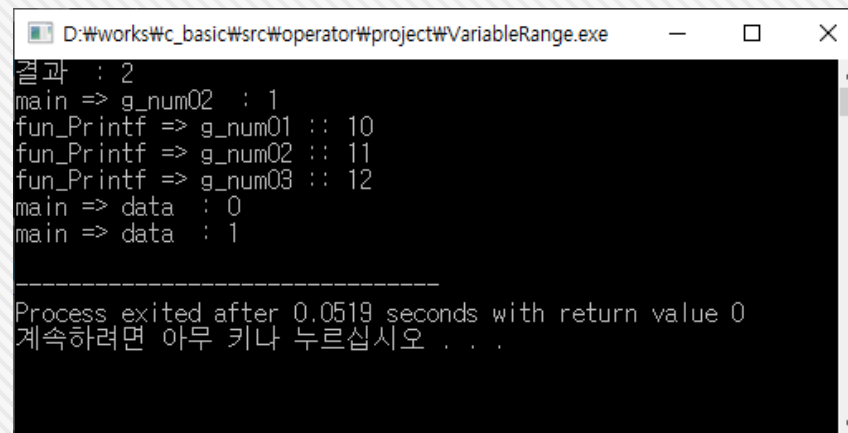
/* int g_num01 = 0; // func.c duerror duplication */
int g_num02 = 1;
/* extren int g_num03 // func.c staic extern*/

int main() {
    funSum(1, 1);
    printf("result : %d\n", result);
    printf("main => g_num02 : %d\n", g_num02);
    fun_Printf();
    funSumAdd();
    funSumAdd();
    return 0;
}

void funSumAdd() {
    static int data = 0;
    printf("main => data : %d\n", data++);
}
```

```
int g_num01 = 10;
static int g_num02 = 11;
static int g_num03 = 12;
void fun_Printf();

void fun_Printf() {
    printf("fun_Printf => g_num01 :: %d\n", g_num01);
    printf("fun_Printf => g_num02 :: %d\n", g_num02);
    printf("fun_Printf => g_num03 :: %d\n", g_num03);
}
```



```
D:\works#c_basic#src#operator#project#VariableRange.exe
결과 : 2
main => g_num02 : 1
fun_Printf => g_num01 :: 10
fun_Printf => g_num02 :: 11
fun_Printf => g_num03 :: 12
main => data : 0
main => data : 1

-----
Process exited after 0.0519 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```

1. 기초 정리

자료형			설명	byte	범위	scanf	printf
정수형	부호 있음 (1 + n)	short	short형 상수	2	-32,768~32,767	&d &o (8 진수) &x (16 진수) %lld (long long) &u (부호없는 십진법 정수)	%d or %i : 부호 있는 십진수 %lld : long long 십진수 %o : 부호 있는 8진수 %u : 부호 없는 십진수 %x : 부호 없는 16진수 (%X)
		int	상수	4	-2,147,483,648~ 2,147,483,647		
		long	long 상수	4	-2,147,483,648~ 2,147,483,647		
		long long	long long 상수	8	-9,223,372,036,854,775,808~ 9,223,372,036,854,775,807		
	부호 없음	unsigned short	부호 없는 short형 상수	2	0 ~ 65535		
		unsigned int	부호 없는 상수	4	0 ~ 4294967295		
		unsigned long	부호 없는 long 상수	4	0 ~ 4294967295		
		unsigned long long	부호 없는 long long 상수	8	0~18,446,744,073,709,551,61	%lld	
문자형	부호 있음	char	문자 or 정수	1	-128~127	&c	%c (문자) , %d (정수)
	부호 없음	unsigned char	문자 or 부호 없는 정수	1	0 ~ 255		
실수형 (부동소수점)		float	단일 정밀도(소수점 7자리)	4	1.17×10의-38승 ~ 3.40×10의38승	&f e, E, f, g, G	%f, %.nf(n:자릿수), 지수 표기 : %e, %E 소수점 이하 자리에 있는 값 까지 : %g, %G
		double	두배 정밀도(소수점 15자리)	8	2.22×10의-308승 ~ 1.79×10의308승	&lf e, E, f, g, G	

2. 연산자

연산자

종류	연산자
대입 연산자	=
산술 연산자	+, -, *, /, &, ++, --
관계 연산자	<, >, <=, >=, ==, !=
논리 연산자	&&, , !
할당 연산자	+=, -=, *=, /=, %=
삼항 연산자	?
비트 연산자	&, , ~, ^, <<, >>

조건문

- If 조건문
- If ~ else ~ 조건문
- 중첩된 조건문
- switch 조건문

반복문

- for 반복문
- while 반복문
- break, continue 제어문

연산자 우선 순위

순위	종류	연산자	연산방향
1	괄호, 배열, 구조체	()(후위증가).[] [후위감소]->	->
2	단항 연산자	(자료형) *(간접) &(주소) !(not) ~(비트) ++ -- +(부호) -(부호) sizeof	<-
3	승제 연산자	* / %	->
4	차감 연산자	+ -	->
5	시프트연산자	<< >>	->
6	비교 연산자	< <= > >=	->
7	등가 연산자	== !=	->
8	비트 연산자 AND	&	->
9	비트 연산자 XOR	^	->
10	비트 연산자 OR		->
11	논리 연산자 AND	&&	->
12	논리 연산자 OR		->
13	조건 연산자	? :	<-
14	대입 연산자	= *= /= += -= %= <<= >>= &= ^= =	<-
15	나열 연산자	,	->