# 3장 헬로 월드

**학습 목표**

C 언어 프로그램의 기본 구조를 알고 있고, 실행되는 순서를 확인할 수 있다.

지금 나의 상태는 \_\_\_\_점이고 이 장을 학습하고 난 뒤 나의 목표는 \_\_\_\_점이다.

1 전혀 모른다

2 이름만 들어봤다

3 어렴풋이 안다

4 설명하기는 어렵지만 사용할 수 있다

5 무엇인지 정확히 알고 설명할 수 있다

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2장에서 구성한 개발환경을 이용하여 내 이름을 화면에 표시하는 프로그램을 작성할 수 있다. |  |  |  |  |  |
| 프로젝트 파일의 기본 구조, 소스 코드의 기본 골격을 이해한다. |  |  |  |  |  |
| 소스 코드를 한 줄씩 실행해 볼 수 있다. |  |  |  |  |  |
|  |  |  |  |  |  |

2장에서는 앞으로 사용할 개발환경을 구성해 보았습니다. 이번 장에서는 이 개발환경을 이용하여 화면에 메시지를 출력하는 간단한 프로그램을 만들어 봅니다. 이를 통해 프로젝트 및 프로그램 소스 코드의 기본 골격을 이해할 수 있습니다. 또한 웹 브라우저에서 동작하는 온라인 컴파일러를 사용하는 방법도 살펴봅니다. 디버깅 도구의 기본적인 사용법을 배운 다음, 코드를 한 줄씩 실행하는 방법을 배워 봅니다.

## 3.1 Hello, TIEs

이전 장에서는 앞으로 C 언어를 배우면서 사용할 개발 환경을 구성했습니다. 이번 장에서는 개발 환경을 사용하여 간단한 프로그램을 만들어 실행해 보면서 프로그램의 구조를 살펴 봅니다. 함께 만들어 볼 프로그램은 콘솔console창에 “Hello, World” 메시지를 출력하는 프로그램입니다. 콘솔 창은 텍스트 명령어를 통해 컴퓨터와 컴퓨터와 직접 소통할 수 있는 창구입니다. 운영체제에 따라 터미널(Terminal) 또는 커맨드 라인(Command Line), 즉 명령행 창으로 불립니다.

NOTE\_

C언어가 개발될 1970년 경에는 우리가 지금 사용하는 것과 같은 높은 수준의 사용자 인터페이스, 즉 GUI 환경이 드물었습니다. 특히 C언어는 애당초 시스템 프로그래밍 언어로 설계되었기 때문에 GUI를 개발하는 데에는 부족한 점이 많습니다. GUI 프로그래밍을 하고 싶다면 대상 GUI 운영체제에 맞는 프레임워크의 구조에 맞게 코드를 작성해야 합니다. 프레임워크framework는 특정 목적에 필요한 프로그램 코드 및 틀을 하나로 묶은 것입니다.

2장에서는 텍스트 파일로 작성된 C 언어 소스 코드가 실행 파일로 만들어지는 과정에 대해 살펴 보았습니다. 보통 여러 개의 C 소스 코드 파일을컴파일한 다음 이를 연결하여 하나의 프로그램을 만들어 냅니다. 하지만 우리는 당분간 하나의 소스 코드 파일에서 하나의 실행 파일을 만들게 될 것입니다. 프로젝트(Project)는 하나의 프로그램을 만드는데 필요한 소스 파일들을 함께 묶은 것입니다. 비주얼 스튜디오는 프로젝트를 기반으로 동작하므로, 반드시 프로젝트를 만들어야 합니다.

이번 장은 처음부터 마지막 단계까지 설명을 따라 직접 해 보는 것이 중요합니다. 때로는 완벽하게 이해되지 않는 부분이 있더라도 끝까지 따라해 본 다음, 다시 각 단계에서 궁금한 점을 자세히 들여다 보는 것이 이해에 도움이 될 것입니다.

1 단계: 비주얼 스튜디오를 실행하면 다음과 같이 ‘새 프로젝트 만들기’ 대화창이 나타납니다. 이 대화창에서는 프로젝트에 사용할 템플릿을 선택합니다. 템플릿(template)이란 무언가를 만들 때 늘 사용되는 기본 틀입니다. 템플릿을 선택하면 이미 정의된 기본 골격에서 프로그래밍을 시작할 수 있습니다. 아쉽게도 C언어 템플릿은 제공되지 않으므로 빈 프로젝트를 선택합니다.

텍스트, 스크린샷, 소프트웨어, 웹 페이지이(가) 표시된 사진

자동 생성된 설명

2 단계: 새로운 프로젝트를 구성하기 위하여 프로젝트 이름과 프로젝트에서 사용할 기본 디렉터리를 지정합니다. 이때 솔루션 파일도 프로젝트 위치에 함께 만들어 집니다. 솔루션(Solution) 파일은 큰 프로그램을 만들 때 관련있는 프로젝트를 모아둔 것으로, .sln 확장자를 사용합니다. 이 책에서 다루는 예제는 프로젝트 마다 솔루션 파일을 만듭니다. 첫번째 프로그래밍 프로젝트이므로 프로젝트 이름을 ‘FirstProject’로, 위치는 프로그래밍 코드를 모아둘 디렉터리로, 솔루션 및 프로젝트는 같은 디렉터리에 만들어 지도록 체크 박스를 켭니다. 그런 다음 ‘만들기’ 버튼을 누릅니다.

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

3단계. 이제 빈 프로젝트가 만들어졌습니다. 솔루션 탐색기 창에서 FirstProject 솔루션에 FirstProject 프로젝트가 들어 있는 것을 볼 수 있습니다. 프로젝트는 기본 골격만 가진, 아무 파일도 들어 있지 않은 빈 상태로 이루어져 있습니다.

솔루션 탐색기는 보통 비주얼 스튜디오 창의 왼쪽에 표시됩니다. 솔루션 탐색기가 보이지 않는다면 보기 > 솔루션 탐색기를 선택하거나 Ctrl + Alt + L을 누릅니다. 이와 같이 솔루션 탐색기에서 프로젝트가 어떤 파일로 이루어지는 지를 확인할 수 있습니다. 특정 파일을 선택한 후 마우스를 더블 클릭하면 편집창에 해당 파일이 열립니다.

프로젝트를 구성하는 파일이 많다면 논리적 가상 폴더를 만들어 관련 파일끼리 묶어서 관리할 수 있습니다. 비주얼 스튜디오는 기본적으로 소스 파일, 헤더 파일, 리소스 파일 폴더를 생성해 줍니다. 이 폴더는 파일 시스템의 폴더와는 아무 관계도 없습니다.

텍스트, 스크린샷, 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명

4단계. 프로그램을 만들기 위한 소스 코드 파일을 프로젝트에 추가해 봅시다. 솔루션 탐색기 창의 소스 파일 폴더를 선택한 후 마우스 오른쪽 버튼을 누릅니다. 팝업 메뉴에서 추가 > 새 항목을 선택합니다. 추가할 새 항목의 템플릿이 표시됩니다. 템플릿 목록에서 ‘C++ 파일(.cpp)’를 선택한 다음, 아래쪽 이름에 “main.c”를 입력한 다음, 추가 버튼을 누릅니다. 소스 파일의 이름은 원하는 대로 사용할 수 있지만, 파일의 확장자는 반드시 ‘.c’여야 합니다. 컴파일러는 파일 확장자에 따라 해당 소스 파일을 어떻게 다루어야 하는지를 알아냅니다.

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

NOTE\_

컴파일러는 소스 파일을 읽어와서 최종 결과인 실행 프로그램을 만들기 위한 자신만의 추상화된 AST 자료구조를 구성합니다. 사용하는 언어에 따라 AST 자료 구조를 구성하는 방식이 달라집니다. 비주얼 스튜디오는 기본적으로 파일의 확장자에 따라 .C는 C언어 소스 파일로, .CPP는 C++언어 소스 파일로, .H는 헤더 파일로 인식하여 해당 프로그래밍 언어의 구문 규칙에 맞게 소스 코드를 이해합니다.

5단계. 프로젝트의 소스 파일 폴더에 main.c파일을 생성하였습니다. 언제든지 솔루션 탐색기에서 파일을 더블 클릭하면오른쪽 편집창에 해당 파일이 열립니다. 편집 창에서는 소스코드를 작성하거나 검토할 수 있습니다.

비주얼 스튜디오는 통합 개발환경이므로, 다른 편집기 프로그램을 사용하지 않고도 편집창에서 소스 코드를 작성하거나 수정할 수 있습니다. 편집창은 코드 작성에 도움이 되는 다양한 기능을 제공합니다. 기본적으로 줄마다 왼쪽에 줄 번호가 표시됩니다. 소스 코드를 입력하면 코드의 의미 단위를 자동으로 인식하여 다른 색상으로 표시됩니다.

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

6단계. 이제 첫번째 소스 코드를 입력해 봅시다. 편집 창에서 코드를 다음과 같이 입력합니다. 줄 번호는 입력하지 않습니다.

코드. main.c 코드 파일

1. /\* TIES-C Programming \*/
2. #include <stdio.h>
3. int main()
4. {
5. printf("Hello, World!\n");
6. return 0;
7. }

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

편집창에서 코드를 입력하면 위와 같이 표시됩니다.코드에 사용된 키워드, 즉 코드의 의미 단위가 사용한 테마에 따라 다른 색상으로 표시되었습니다.

우리가 입력한 소스 코드는 C 프로그래밍에 대한 최초의 책에 소개된, C 언어로 작성된 가장 오래된 코드입니다. 보통 새로운 프로그래밍 언어를 배울 때 이와 같이 콘솔창에 “Hello, World”문장을 출력하는 것에서 시작합니다.

편집기 창에서 코드를 입력한 다음, 실행 파일을 만들어 코드를 실행해 봅시다. 실행 파일을 만들려면 빌드 과정을 거쳐야 합니다. Ctrl+B 또는 빌드> 솔루션 빌드를 선택합니다. 프로젝트 빌드 설정에 따라 프로젝트 폴더의 Debug 폴더 또는 Release폴더에 실행파일이 만들어 집니다.

프로젝트 빌드 설정은 상단 툴바의 콤보 박스에서 변경할 수 있습니다. 빌드 설정에서는 디버그 빌드를 만들 것인지 릴리즈 빌드를 만들 것인지, 어떤 CPU를 기반으로 동작하는 실행파일을 만들지 등을 지정합니다.

[그림 추가 필요 – 빌드 툴바: 빌드 종류, CPU]

정상적으로 빌드가 수행되지 않고 오류가 발생했다면 입력한 소스 코드가 예제 코드와 정확히 일치하는지를 확인합니다.

NOTE\_

빌드 과정을 거쳐 만들어진 실행 프로그램은 다른 컴퓨터로 복사한 후 실행해도 똑같이 동작합니다. 다만 실행파일의 크기를 줄이기 위해 공통적인 실행 코드를 운영체제에 하나만 설치해 두고 함께 사용합니다. 이를 C 런타임 라이브러리라고 부릅니다. 일반적으로 C 런타임 라이브러리가 운영체제와 함께 설치되어 있습니다만, 설치되어 있지 않다면 따로 설치해야 합니다. 디버그용 빌드를 만들었을 경우 디버그용 C 런타임 라이브러리가 필요한데, 보통 설치되어 있지 않습니다. 또한 디버그 빌드에는 개발 과정에서 필요한 부가 정보가 실행파일에 포함되어 있기 때문에 파일 크기도 릴리즈 빌드에 비해 크고, 느립니다. 따라서 개발한 프로그램을 다른 컴퓨터에 배포할 때에는 릴리즈 빌드를 전달해야 합니다.

7단계. 만들어진 프로그램을 실행해봅시다. 메뉴에서 디버그 > 디버깅 시작을 선택하거나 F5를 누르면 우리가 만든 프로그램이 실행된 결과를 볼 수 있습니다. 제대로 동작했다면 화면에 “Hello, World!” 문구가 화면에 표시됩니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

물론 탐색기에서 프로젝트의 Release 폴더에 생긴 FirstProject.exe를 실행해도 됩니다. 이 경우 프로그램이 종료되면 콘솔창이 자동으로 닫힙니다. 따라서 명령행 프롬프트를 실행하여 해당 실행파일을 실행해 주어야 합니다.

## 3.2 코드를 한 줄씩 살펴보기.

프로젝트는 여러 개의 C 언어 코드 파일로 이루어집니다.각 파일은 크게 소스 파일과 헤더 파일로 나누어 집니다. 보통 관련있는 파일들을 같은 경로에 저장합니다. 하지만 실제 경로와 상관없이 파일의 목적에 따라 논리적인 그룹으로 묶기도 합니다. 솔루션 탐색기에서는 논리적 폴더를 생성하여 언제든 관련있는 파일들을 하나의 그룹으로 관리할 수 있습니다.

먼저 기본적인 코드 규칙부터 살펴보도록 합시다.

C 컴파일러는 소스 코드를 읽을 때 명령어 사이에 있는 빈 칸을 나타내는 공백 문자나 탭 문자를 신경쓰지 않습니다. 즉, 빈 줄이나 들여쓰기가 있더라도 프로그램의 실행에는 영향을 미치지 않습니다. 하지만 개발자는 들여쓰기나 한 줄 띄우기를 통해 시각적으로 코드를 구분해 줌으로써 코드를 쉽게 읽고 이해할 수 있도록 도와줍니다. 읽기 좋은 코드를 가독성readability이 좋은 코드 또는 가독성이 높은 코드라고 부릅니다.

C언어에서는 명령문의 끝을 세미콜론(‘;’)으로 표시합니다. 보통 값을 다루는 선언문과 대입문, 함수 호출문, 조건문이나 반복문 내의 실행문 마다 세미콜론을 붙입니다. 다만, 함수 선언부와 조건문의 실행 조건에는 세미콜론을 붙이지 않습니다.

낯선 용어들이 툭 튀어나와서 어렵게 느껴질 수 있습니다. 하지만 이 책에서 조금씩 자세히 살펴볼 테니 걱정하지 마세요.

지금부터 우리가 작성한 C 소스 코드를 한 줄씩 상세히 살펴 보겠습니다.

첫번째 줄은 /\* 으로 시작해서 \*/으로 끝이 납니다. 이런 코드를 주석comment이라 부르는데, 소스 코드를 읽는 사람을 위해 코드에 표시해 둔 추가 설명입니다. 보통 코드의 기능에 대한 설명을 덧붙이거나 기존 코드의 일부를 잠시 사용하지 않을 때, 또는 코드 문서화 프로그램에 필요한 정보를 전달하기 위하여 주석을 사용합니다. 컴파일러는 코드 파일을 다룰 때 주석에 대해 별도의 처리 과정을 수행하지 않으므로, 프로그램의 동작에는 아무런 영향을 미치지 않습니다.

주석은 한 줄 주석과 여러 줄 주석, 두가지 형태로 사용합니다. 한 줄 주석은 ‘//’를 사용하여, 해당 위치부터 그 줄이 끝나는 부분까지, 나머지 부분을 주석으로 만듭니다.

// 한줄 주석 - 화면에 Hello, TIES 문장을 출력한다.

printf(“Hello, TIES”);

printf(“Hello, TIES”); // 화면에 Hello, TIES 문장을 출력한다.

여러 줄 주석은 ‘/\*’로 시작하며, 이후 처음 만나는‘\*/’에서 끝나는 영역까지를 주석으로 표시합니다. 물론 이 방식으로도 예제 코드와 같이 한 줄 주석처럼 사용할 수 있습니다.

/\* 한줄로 된 주석문을 다음과 같이 작성할 수 있다.\*/

/\* 여러 줄로 된 주석문을

다음과 같이 작성할 수 있다. \*/

예제 코드에서는 주석을 사용하여 TIE 시리즈에 소개된 예제임을 코드를 읽는 사람에게 알려주고 있습니다.

두번째 줄은 빈 줄입니다. 빈 줄을 추가함으로써 첫번째 줄과 세번째 줄이 서로 다른 역할을 수행하고 있음을 알려줍니다. 4행과 8행 역시 코드를 구분해 주는 용도로 쓰였습니다만, 가독성을 좋게 만들 뿐 코드의 동작에는 아무런 영향을 미치지 않습니다.

세번째 줄에서는 헤더 파일을 포함합니다. #include 는 전처리 지시자로, 지정된 별도 파일을 해당 소스 파일의 위치에 포함시킵니다. 즉, 해당 소스 파일을 컴파일하기 전에 지정된 파일의 내용이 해당 코드의 위치에 삽입되는 전처리 과정을 거친 후, 이 결과 코드를 컴파일하게 됩니다. 보통 헤더 파일을 포함할 때 사용합니다. 헤더 파일에는 함수를 어떻게 사용하는 것인지에 대한 정보가 들어있습니다.

예제 파일에서는 stdio.h 파일을 포함합니다. 헤더 파일은 보통 .h 확장자를 사용합니다. 파일명 앞뒤에 꺽쇠(<>) 또는 큰따옴표(“”)를 붙여줍니다. 꺽쇠는 컴파일러가 지정된 디렉터리에서 해당 파일을 찾기 때문에 표준 라이브러리와 관련된 헤더 파일을 포함할 때 사용하며, 큰따옴표는 현재 디렉터리 또는 지정된 경로에서 해당 파일을 찾기 때문에 개발자가 만든 헤더 파일을 포함할 때 사용합니다.

stdio.h 헤더 파일에는 printf 함수를 비롯하여 표준 라이브러리에서 제공하는 표준 입출력과 관련된 함수들이 선언되어 있습니다. 자주 사용하는 표준 라이브러리 헤더파일 중 몇가지를 소개하면 다음과 같습니다.

|  |  |
| --- | --- |
| 헤더 파일명 | 설명 |
| stdio.h | 표준 입출력 함수 |
| stdlib.h | 표준 일반 유틸리티 함수. 메모리 관리, 문자열 변환, 알고리즘에 관련된 함수 |
| string.h | 문자열을 다루는 함수 |
| time.h | 시간과 날짜를 다루는 함수 |
| thread.h | 스레드 관련 함수. C11 표준에 추가됨 |

NOTE\_

헤더 파일은 왜 만들어 졌고, 왜 여러 개로 나누었을까요? 이렇게 헤더파일을 나눠두면 여러 소스 파일에서 함께 사용하는 함수나 매크로 등을 재사용할 수 있을 뿐만 아니라 코드가 긴 것보다 여러 개의 파일로 나눠서 관리하는 것이 쉽기 때문입니다.

C언어에서 함수function는 연관된 작업들을 하나의 기본 실행 단위로 묶은 것입니다. 함수는 입력값을 받아서 어떤 동작을 수행하고, 함수를 실행한 코드에게 결괏값을 되돌려줍니다. 이때 함수가 수행할 동작은 중괄호, 즉 ‘{‘와 ‘}’ 사이에 작성하며, 코드는 위에서 아래로 차례대로 실행됩니다.

다음은 함수의 기본 형태입니다.

함수가\_되돌려줄\_결괏값의\_형태 함수\_이름(함수에\_전달할\_입력값\_목록)

{

함수에서\_실행할\_동작\_1;

함수에서\_실행할\_동작\_2;

return 함수가\_되돌려줄\_값

}

이렇게 정의된 함수를 사용할 때에는 다음과 같은 형태로 사용합니다. 함수를 사용하는

함수\_이름(함수에\_전달할\_입력값\_목록);

함수\_결괏값을\_저장할\_공간 = 함수\_이름(함수에\_전달할\_입력값\_목록);

따라서 위 예제 코드는 다음과 같이 이해할 수 있습니다.

* 이 함수의 이름은 main 입니다.
* 이 함수의 입력값은 없습니다. 즉, 아무 값도 전달하지 않습니다.
* 이 함수의 출력값은 정수(int)값입니다.
* 이 함수는 printf 함수를 호출합니다. printf 함수에 “Hello, World!” 문자열을 단일 입력값으로 전달합니다.
* 함수가 끝나면 0값을 되돌려 줍니다.

소스 코드는 기본적으로 위에서 아래로 작성된 코드를 한 줄씩 실행하면서 진행합니다. 코드가 수행할 작업 덩어리를 함수라고 볼 수 있습니다. 함수는 뒤에서 보다 자세히 다룹니다.

C 프로그램은 이와 같이 여러 개의 함수로 구성합니다. 그러면 프로그램이 실행되었을 때 가장 먼저 실행해야 하는 함수는 무엇일까요? 바로 main 입니다. 그래서 main함수를 프로그램의 진입점entry point이라고 부릅니다. 즉, 프로그램이 실행되면 main함수 내에 작성된 코드 동작을 시작하며, main함수 내에 정의된 코드를 모두 수행하고 나면 프로그램은 종료됩니다. 따라서 C 프로그램을 작성할 때 반드시 작성해야 하는 단 하나의 함수가 있다면 main함수여야 합니다.

C 프로그램을 작성할 때 반드시 작성해야 하는 단 하나의 함수가 있다면 main함수여야 합니다. main함수가 없다면 컴파일러는 어떤 코드부터 작업을 수행해야 할 것인지를 알수 없습니다.

[그림 추가 예정 - main 함수 시작과 끝]

C 소스 코드에서는 각 명령 구문을 구분하기 위하여 세미콜론(;)을 사용합니다. 즉, 단위 동작을 수행하는 코드마다 세미콜론을 붙여 주어야 합니다.

NOTE\_ 구문(statement)과 표현식(expression)

구문/표현식에 대한 설명.. 나중에 추가 하겠습니다.

프로그램이 수행할 동작을 개발자는 코드로 작성합니다. 그 동작들 중 공통적으로 자주 사용하는 기능은 컴파일러에서 미리 구현한 상태, 즉 바로 사용할 수 있는 상태로 제공합니다. 이와 같이 개발자들이 자주 사용하는 기능을 미리 함수로 구현하여 모은 것이 표준 라이브러리(standard library)입니다.

표준 라이브러리에서 제공하는 여러 함수 중 printf 함수는 화면에 문자열을 표시하는 동작을 수행합니다. printf(“출력 문자열");와 같은 코드를 작성하면 콘솔 창에 “출력 문자열”을 표시합니다. 문자열을 표시할 때에는 큰 따옴표를 사용합니다.

#include <stdio.h>

main 함수의 마지막에 return 구문이 사용되었습니다. return 구문을 만나면 함수를 호출한 코드에게 return 뒤에 적힌 값을 반환합니다. main함수의 반환값은 정수형으로 , 보통 함수의 실행결과를 알려주기 위해 사용하는 코드를 반환한다.

main함수는 다른 함수에서 호출한 함수가 아니므로, 운영체제에 해당 값을 알려줍니다. 보통 미리 정의된 값을 반환하는데, 일반적으로 아무 문제 없이 동작하였을 경우 0값을 되돌려 줍니다.

지금까지 예제 프로그램의 수행 동작과 관련된 코드를 살펴보았습니다.

## 3.3 코드를 한줄씩 실행해 보기

지금까지는 실행 파일을 만들어 실행해 보았습니다만, 개발을 하다 보면 의도한 대로 동작하지 않을 수 있습니다. 프로그램의 오류를 찾아내고 수정하는 과정을 디버깅(debugging)이라고 부릅니다.

디버깅을 할 때 코드를 한 줄씩 실행해 보면서 프로그램의 흐름이 원하는대로 흘러 가는지, 원하는 작업을 수행하는지 확인해 볼 수 있다면 정말 편리하지 않을까요? 이를 도와주는 도구가 바로 디버거debugger입니다.하지만 실행파일과 해당 동작의 실제 코드가 어느 부분인지를 연결해 주는 정보가 있어야 합니다. 이 정보를 가진 실행파일을 만들기 위해서는 대상 빌드를 디버그로 설정해야 합니다.

비주얼 스튜디오의 툴바의 빌드 구성을 Debug로 설정합니다. 이렇게 만들어진 디버그 빌드는 컴파일러가 실행속도를 높이기 위하여 수행하는 최적화 과정을 거치지 않고, 디버깅에 필요한 심볼 정보를 포함하고 있기 때문에 실행파일의 크기가 릴리즈 빌드에 비해 더 클 뿐만 아니라 메모리도 많이 사용하고 속도도 느립니다. 따라서개발하는 과정에서는 디버그 빌드를 사용하고, 완성된 프로그램을 사용할 때에는 Release를 선택하여 실행파일을 만듭니다.

먼저 비주얼 스튜디오에서 디버그 > 디버깅 시작, 또는 F5키를 눌러 디버깅을 시작합니다. 이전과 마찬가지로 프로그램이 실행되어 “Hello, World!”메시지가 화면에 출력되고 프로그램은 종료됩니다. 디버깅을 하려면 먼저 자세히 살펴볼 코드에 중단점(break point)을 설정해야 합니다.

예제에서는 7번 줄에 중단점을 설정합니다. 코드 편집창에서 7번 줄로 커서를 옮긴 다음, F9 키를 누릅니다. 줄번호 옆에 빨간 원 모양이 표시됩니다.비주얼 스튜디오는 디버그 빌드를 실행하면서 디버거에 연결합니다. 디버그 빌드 실행 파일은 동작하다가 중단점을 만나면 해당 줄 코드의 실행을 잠시 멈춥니다. 코드 실행이 멈춰진 상태에서 호출 스택이나 메모리 검사 창을 통해 프로그램이 어떤 상태인지를 확인할 수 있습니다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

예제에서 아직 데이터를 다루지 않기 때문에, 데이터가 어떤 상태인지를 확인하기 보다는코드를 한 줄씩 실행하는 방법을 살펴보겠습니다. 다시 F5키를 눌러 디버깅을 시작합니다. 이번에는 콘솔창에서 프로그램이 실행된 다음, 7번째 줄에 화살표가 표시되면서 계속 코드가 실행되지 않고 멈춰있습니다.멈춰진 상태에서는 현재 메모리의 상태는 어떤지, 어떤 값이 저장되어 있는지를 확인해 볼 수 있습니다.

F10키를 누르면 다음에 실행할 코드 구문을실행합니다. 따라서 printf 구문이 실행되면서 콘솔창에 “Hello, TIES!”메시지가 표시되고화살표가 9번째 줄로 옮겨집니다.

왜 7번째 줄 다음에 9번째 줄이 실행될까요? 앞서 설명한 것처럼 C컴파일러는 코드에 있는 빈 줄은 따로 처리하지 않습니다. 따라서 다음 실행 구문이 있는 9번째 줄로실행 지점이 옮겨집니다. 컴파일러는 빈 줄을 무시하지만, 개발자들은 빈 줄을 사용하여 코드의 맥락을 표현해 줍니다. 코드 실행 블럭의 의도가 변경되는 부분에서 적절히 빈 줄을 추가하거나 줄 앞부분에 공백을 추가함으로써 코드 논리 영역을 시각적으로 나눠 주면 훨씬 쉽게 코드를 이해할 수 있습니다.

다시 한번 F10 키를 누르면 “return 0;” 구문이 실행되면서 main함수의 실행이 끝나고, 프로그램의 실행은 끝나게 됩니다.

디버깅은 프로그램을 만드는 과정에서 어쩔 수 없이 마주하게 되는 과정입니다. 프로그램이 의도한 동작을 제대로 수행할 때까지 디버거 도구를 사용하여 코드를 확인하고 수정하는 과정을 거치게 됩니다. 디버깅 도구는 특정 코드까지 실행하기,함수 단위로 실행하기, 변수값 확인하기 등 훨씬 다양한 기능을 가지고 있습니다. 나중에 함수를 배운 다음 디버거를 사용하는 방법을 다시 한번 살펴보도록 하겠습니다.

TIPS\_

맥이나 리눅스에서 C언어를 공부하는 경우라면 gdb(GNU debugger)를 사용합니다. 보통 IDE에는 코드 작성 도구, 컴파일러 도구와 디버깅 도구가 모두 통합되어 하나의 도구에서 매끄럽게 원하는 도구를 사용할 수 있습니다.

## 3.4 온라인 C 컴파일러로 실행해 보기

지금까지는 비주얼 스튜디오가 설치된 컴퓨터에서 C코드를 작성한 후 실행해 보았습니다. 이번에는 웹 브라우저에서 온라인 C 컴파일러에 접근하여 코드를 작성한 후 실행해 보겠습니다.

온라인 C 컴파일러는 기본적으로 서버에서 실행된 결과만 웹 브라우저로 표시해 주므로 실행파일을 직접 만들 수는 없습니다. 다만 특별한 도구를 설치하지 않아도 웹 브라우저만 있으면 C언어를 배우는 과정에서 손쉽게 코드를 작성, 실행해 볼 수 있습니다.

여러 종류의 온라인 컴파일러가 존재하지만, 이번에는 온라인 GDB를 사용합니다. 특별한 차이는 없으므로 자신의 마음에 드는 온라인 컴파일러를 사용하면 됩니다. 온라인 서비스이므로 같은 서비스일지라도 겉모습은 사용 시점에 따라 조금씩 달라질 수 있습니다.

브라우저를 실행하여 주소 창에 https://onlinegdb.com 을 입력합니다. 다음과 같은 편집창이 표시됩니다.

텍스트, 소프트웨어, 멀티미디어 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명

오른쪽 상단에 있는 Language 콤보 리스트에서 C를 선택합니다. 그러면 편집창의 파일명이 자동으로 main.c로 변경됩니다.

편집창에는 이미 우리가 작성한 예제와 비슷한 코드가 채워져 있습니다. 온라인 컴파일러 역시 편집 창에 줄 번호가 표시되고, 키워드에 따라 다양한 색상으로 표시되어 코드를 쉽게 이해할 수 있습니다. 이번에는 코드를 읽어 봅시다.

기존 예제 코드와 비슷한 모양새를 갖추고 있습니다. 13번째 줄에 출력되는 문자열을 다른 문자열로 바꿔실행해 봅시다.

기존: printf(“Hello World”);

변경: printf(“Hello, TIES!”);

텍스트, 소프트웨어, 멀티미디어 소프트웨어, 그래픽 소프트웨어이(가) 표시된 사진

자동 생성된 설명

편집창 윗쪽 버튼 모음에서Run(실행) 버튼을 누릅니다. 그러면 편집창에 작성된 코드는 서버에서 컴파일되기 시작합니다. 아무런 문제없이 컴파일과정을 마치면 서버에서 실행 프로그램의 실행 결과가 브라우저의 편집창 아래쪽 결과창에 표시됩니다.

온라인 컴파일러에서도 디버그 기능을 사용할 수 있습니다. 다만 우리가 사용하는 온라인 컴파일러는 GDBGNU Debugger라는 별도의 디버거 프로그램을 사용해야 합니다. 편집창 윗쪽 버튼 모음에서 Debug버튼을 누릅니다. 이번에는 실행결과가 표시되었던 화면에 GDB 메시지가 표시되고, 편집창 오른쪽에는 디버깅 과정에 필요한 패널창들이 표시됩니다.

텍스트, 스크린샷, 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명

비주얼스튜디오에서는 중단점을 마우스 또는 단축키로 설정하였습니다만, GDB에서는 중단점을 설정하는 명령을 실행해야 합니다. 위 코드에서 main.c 파일의 13번째 줄의 코드에 중단점을 설정해 보겠습니다. 해당 줄에 중단점을 설정하기 위하여 “break main.c:13” 명령을 입력합니다.

Reading symbols from a.out...

(gdb) break main.c:13

Breakpoint 1 at 0x1151: file main.c, line 13.

중단점이 설정되었으므로 디버그 모드로 컴파일된 프로그램을 실행해 봅니다. “run”명령을 입력합니다. 그러면 프로그램이 실행되어 중단점을 만날 때까지 실행되고, 중단점이 설정된 코드가 화면에 표시됩니다.

(gdb) run

Starting program: /home/a.out

[Thread debugging using libthread\_db enabled]

Using host libthread\_db library "/lib/x86\_64-linux-gnu/libthread\_db.so.1".

Breakpoint 1, main () at main.c:13

13 printf("Hello, TIES!");

(gdb)

중단점이 설정된 코드 부분을 실행하려면 “next”명령을 입력합니다. 이제 한 줄씩 코드가 실행되면서 프로그램이 종료될 때까지의 상태를 확인할 수 있습니다.

(gdb) next

15 return 0;

(gdb) next

16 }

(gdb) next

\_\_libc\_start\_call\_main (main=main@entry=0x555555555149 <main>, argc=argc@entry=1, argv=argv@entry=0x7fffffffec58) at ../sysdeps/nptl/libc\_start\_call\_main.h:74

74 ../sysdeps/nptl/libc\_start\_call\_main.h: No such file or directory.

(gdb)

GDB는 다양한 명령을 통해 여러 동작을 수행할 수 있습니다. 하지만 그 사용법을 익히는 것은 이 책의 범위를 벗어나므로 더 이상 다루지는 않습니다.

이 책의 대부분의 예제를 익힐 때에는 온라인 컴파일러를 사용해도 괜찮습니다. 다만 회원 가입을 하지 않으면 브라우저를 종료하면 작성한 코드가 사라져 버리고, 실행 결과는 확인할 수 있지만 실행 파일을 만들 수 없습니다. 또한 디버깅 과정도 자신의 컴퓨터에 설치된 개발환경에 비해 불편합니다. 따라서 이 책에서는 온라인 컴파일러로도 코드를 익힐 수 있지만, 비주얼 스튜디오를 설치한 환경을 기준으로 설명합니다.

## 3.5 요약

이번 장에서는 매우 간단한 콘솔 프로그램을 만들어 보면서 프로그램의 구조와 개발환경의 사용법에 대해 살펴보았습니다.

● C 프로그램에는 반드시 main함수가 있어야 합니다.

● 프로그램은 main함수에 작성된 코드를 순서대로 실행하며, main함수의 실행을 마치면 프로그램은 종료됩니다.

● C 코드에서 구문은 세미콜론(;)으로 끝납니다.

● 주석은 코드의 이해를 돕기 위해 코드에 남겨두는 문장입니다. 적절한 주석은 코드의 이해를 돕습니다.

● 디버그는 개발자가 의도한 대로 코드가 동작하는 지를 확인하기 위한 도구입니다. 중단점을 설정하여 특정 코드가 실행된 순간에 프로그램의 실행 상태를 확인해 볼 수 있습니다.

## 심화학습

### 과제

1. 자신의 이름을 한글로 출력하는 프로그램을 만들어 봅시다.

## 도움자료

GNU GDB 매뉴얼: http://korea.gnu.org/manual/release/gdb/gdb.html

—-------------------------------

혹시나 나중을 위해 남겨놓은 초안

윈도우 탐색기에서 exe 파일을 더블 클릭하여 실행할 수도 있지만, 이 경우 출력 콘솔 창이 실행 완료와 함께 사라지므로, 가급적 명령행 도구를 실행한 다음 프로그램을 실행하는 것이 좋습니다.

다음과 같은 결과 가 표시됩니다.

옵션 >

도구 >

솔루션 디렉터리 설명.

디버그/릴리즈 빌드 - 런타임으로 설명하기.

이번에는 온라인 컴파일러에서 실행해 본다. (한번만 설명)

이제 코드를 한줄씩 이해해 봅시다.

Main함수. C언어 프로그램의 시작과 끝.

main함수는 두가지 버전이 있지만, 당분간은 입력이 없는

지시문은 한번에 하나씩. 끝날때마다 세미콜론을 찍어 준다.

printf는 화면에 출력을 하기 위한 함수. stdio.h라는 파일에 정의되어 있음.

왜 헤더 파일을 따로 정의했을까.. 레고 블럭도 연결할 수있는 부분을 미리 알아야 연결할 수 있으니까. 그래서 함수의 모양새를 미리 알려줘야 한다. 그래서 매번 소스코드를 컴파일할 때 알려줄 수 없으니 따로 빼서 #include 로 포함시킨 것.

Retrun

주석. 코드를 작성하는 것은 컴퓨터에 시킬 작업 지시문. 그 작업을 변경하는 것은 결국 사람. 코드를 읽고 바로 의도를 이해할 수 있으면 가장 좋지만 그렇지 않을 경우에는 추가 설명이 필요합니다.

/\* \*/

//

“설명이 필요한 개그는 실패한 개그이다"

하지만 모호할 때는 설명이 필요합니다. -> 클린 코드

빈줄은 실행 파일을 만드는 데는 의미가 없지만, 시각적으로 코드를 분리시켜 주므로 적절히 빈줄로 영역을 나눠 주자.

함수는 작은 함수들로 이루어 진다.

디버깅.

이제 잠시 책을 기존 코드에 “안녕하세요"를 출력해 보자…. 한글도 출력되는 것만 확인. 문자열 인코딩 이런건 나중에 이야기 하자.

한줄씩 실행 -> 로컬 변수 살펴 보기

한줄씩 실행하다간 원하는 부분까지 가다가 죽을꺼야,. 중단점 설정해서 원하는 부분까지 가서 좀 더 살펴보자.

Step over같은 것은 용어만 설명하는 수준에서 넘어갈 것.

—-

드디어 첫번쨰 프로그램을 실행해 보았다. 이제 이 프로그램 골격을 바탕으로 계속 확장시켜 보자.

컴파일러는 개별 소스코드 단위로 동작을 이해하여 기계가 수행할 수 있는 코드로 변환하므로, 표준 라이브러리를 비롯하여 해당 파일에 정의되지 않은 함수에 대해서는 어떤 식으로 사용되는지를 알 수 없습니다. 이 문제를 해결하기 위하여 사용하는 함수의 모양을 선언해 주어야 하는데, 이와 같은 함수 선언을 모아둔 파일이 헤더 파일입니다. 확장자는 .h를 사용하며, 특정 함수를 사용하기 위해서는 include 문을 사용하여 적절한 헤더 파일을 포함시켜주어야 합니다.