Hanbit eBook

Realtime 64

BACK TO THE BASIC

뿌리부터 이해하는 C 언어 Vol II

이승미, 이희정 지음

한빛미디어 Hanbit Media, Inc.

BACK TO THE BASIC

뿌리부터 이해하는 C 언어 Vol.II

BACK TO THE BASIC **뿌리부터 이해하는 C 언어 Vol. II**

초판발행 2014년 5월 30일

지은이 이승미, 이희정 / **펴낸이** 김태헌
펴낸곳 한빛미디어(주) / **주소** 서울시 마포구 양화로 7길 83 한빛미디어(주) IT출판부
전화 02-325-5544 / **팩스** 02-336-7124
등록 1999년 6월 24일 제10-1779호
ISBN 978-89-6848-715-6 15000 / **정가** 12,000원

책임편집 배용석 / **기획** 김병희 / **편집** 안선화
디자인 표지 여동일, 내지 스튜디오 [밈], 조판 김현미
영업 김형진, 김진불, 조유미 / **마케팅** 박상용, 서은옥, 김옥현

이 책에 대한 의견이나 오탈자 및 잘못된 내용에 대한 수정 정보는 한빛미디어(주)의 홈페이지나 아래 이메일로 알려주십시오.
한빛미디어 홈페이지 www.hanbit.co.kr / **이메일** ask@hanbit.co.kr

Published by HANBIT Media, Inc. Printed in Korea

지금 하지 않으면 할 수 없는 일이 있습니다.
책으로 펴내고 싶은 아이디어나 원고를 메일(ebookwriter@hanbit.co.kr)로 보내주세요.
한빛미디어(주)는 여러분의 소중한 경험과 지식을 기다리고 있습니다.

저자 소개

저자 **이승미**

서울여자대학교 컴퓨터공학과/정보보호공학과를 졸업했다. 모바일 관련 벤처기업에서 8년간 근무하며 C와 C++ 기반의 브라우저 개발과 국내 및 해외 모바일 솔루션 상용화에 참여했다. 모바일 환경이 안드로이드로 옮겨가는 시기에 안드로이드 환경에 적합한 오피스 관련 제품군 상용화에 참여하였으며, 현재는 작은 모바일 앱 개발 업체인 '소프트씨드'의 대표이자 개발자로 근무 중이다.

- seungmil@gmail.com | http://androbook.tistory.com

저자 **이희정**

국민대학교 컴퓨터응용학과 졸업 후 한양대학교 컴퓨터공학과에서 석사과정을 마쳤다. 모바일 관련 벤처기업에서 7년간 근무하며 C와 C++ 기반의 브라우저 개발을 비롯해 국내뿐만 아니라 동남아시아 여러 국가의 모바일 솔루션 상용화에도 참여했다. 현재는 C와 C++, 웹 기술들에 관심이 있으며 다양한 분야의 컴퓨터 서적을 번역하는 등 프리랜서로 활동 중이다.

- heejoung80@gmail.com | http://ichbbol.blogspot.com

스페셜 코드 에디터_ **김태희**

현재 캐나다 몬트리올에 있는 Warner Bros Games Montreal 서버 팀의 팀장으로 일하며 크고 작은 온라인 게임을 제작하고 있다. 이 책의 예제 리뷰에 참여했다.

- steeple@naver.com | http://blog.eeodl.com

저자 서문

같은 회사를 7년 정도 다니고 비슷한 즈음 그 회사를 그만둔 인연으로 어찌어찌 함께 C 언어에 관한 책을 써 보기로 했을 때, 과연 우리가 책을 쓸만한 그릇이 되나 하고 잠시 고민했었다. 생초짜 개발자 시절부터 무시무시한 프로젝트에 눌려 꾸역꾸역 버텨내긴 했지만, 남에게 알려줄 만한 지식을 쌓아놓긴 했는지 스스로 의심스러웠기 때문에. 하지만 둘 다 뭔가 결정하는 데 오래 고민하는 스타일은 못 돼서, "일단 해보자"고 결정을 내린 후 그해 겨울 줄곧 커피숍에서 만나 의논해가며 나름 즐겁게 원고를 진행했다. 책을 써 내려가며 우스갯소리로 "우리가 C를 하긴 했구나"라는 말을 했다. 뭣도 모르고 개발을 시작해서 여전히 아는 것은 하나도 없는 것 같은데, 막상 누군가에게 C를 알려줘야겠다고 생각하니 전해 줄 말이 너무 많았다.

원고를 완성하고 사실 우리의 원고가 묻힐지도 모르겠다는 생각을 했다. 다행히(?) 나름 열심히 만든 원고가 책이 되어 나온다고 하니 심장이 덜컹거린다. "제대로 책을 썼나?"라는 걱정이 앞선다. 이 책이 개발자로서의 첫걸음을 내딛는 많은 이들에게 조금이나마 도움이 되길 바라는 마음이다.

Thanks to

승미

첫 입사 이후 줄곧 무시무시한 프로젝트를 통해 '스파르타'식으로 저를 이끌어주신 구용구 수석님(가끔 저를 비웃다가 눈물 흘리신 건 잊어드릴게요), 8년여 세월 동안 같은 회사에서 개발자로서 서로 보탬이 되었던 동료들(은경, 미주, 정훈, 광균……, 너무 많아 생략), 무심한 파트장(?) 만나서 고생한 우리 애기들(선욱, 범삼, 상훈, 기현, 관구, 애기라곤 할 수 없지만 함께 힘들게 일했던 재성, 계형, 동건)을 비롯해서 함께 일했던 모든 분께 감사드립니다. 생에 첫 집필에 함께 기뻐해준 우리 가족들, 친구들 감사드립니다. 언제나 내 삶에 기쁨에 되어주는 우리 조카들(예원, 예나), 이모가 무지 사랑해. 책을 출판할 기회를 만들어주신 엄진영 님, 출판에 힘써주신 김병희 님 및 한빛미디어의 많은 분께 감사드립니다. 함께 원고를 집필하느라 고생한 희정, 정말 수고했어.

희정

집필하는 동안 많은 지원을 해주신 가족들께 감사드립니다. 저에게 끝없는 용기를 불어넣어 준 때이님 감사합니다. 출판 업계에 뛰어들게 해주신 김병부 님께 감사드립니다. 그리고 이 책을 출판하기까지 정말 많이 도와주신 한빛미디어의 김병희 님과 스마트미디어팀 여러분께 감사드립니다. 함께 책 쓰고 먹으러 다니느라 고생한 승미 언니, '소프트씨드' 성공하길 빌어요.

대상 독자 및 예제 파일

초급 | 초중급 | **중급** | 중고급 | 고급

이 책은 '프로그래밍을 해보자'에 초점이 맞춰져 있다.

C 프로그래밍을 시작하기 위해 알아야 할 것은 그리 많지 않다. 필자는 기본적으로 컴퓨터를 열어서 프로그램을 설치하고, 키보드를 사용해 글자를 타이핑할 수 있는 사람이라면 모두 프로그래밍을 할 수 있다고 본다. C 프로그래밍을 할 수 있는 프로그램을 자신의 컴퓨터에 설치하고, 그 프로그램을 실행해서 몇 줄의 코드를 작성한 뒤, 컴파일을 시작하는 버튼을 클릭한다. 그리고 실행 버튼을 클릭하여 방금 작성한 프로그램을 눈으로 확인한다. 이 과정 자체가 프로그래밍이며, 위에 단 몇 줄로 적은 바와 같이 이것은 그다지 어려운 과정이 아니다.

이러한 과정을 따라해보고 코드라는 것을 작성하기 위해 필요한 기초적인 것들을 배우면서 몇 줄의 코드를 더 작성해본다. 프로그래밍을 시작하기 위해 모든 이론을 완벽히 알고 있을 필요는 없다. 어려운 부분이 있다면 일단 넘어가자. 그 모든 부분을 알아야만 프로그래밍을 할 수 있는 것은 아니다. 일단 프로그래밍이 어떤 것인지 경험해보길 권한다.

영어와 한국어 등 세상에 서로 다른 언어가 많듯이, 컴퓨터가 알아듣는 언어에도 여러 가지가 있다. C, C++, Java 등이 컴퓨터가 알아들을 수 있는 언어이며, 이 책에서는 이들 중 C라는 언어를 사용해서 컴퓨터와 소통하는 방법을 익혀갈 것이다.

영어를 처음 배울 때 "I'm a girl."을 배우고 "I'm a beautiful girl."을 배우듯, 초반에 일단 프로그래밍이란 것을 경험해보고, 각 단락 단락에서 기본적인 코드에 추가적으로 붙여나갈 수 있는 것들에 무엇이 있는지를 확인할 것이다. 영어선생님은 영어하는 방법을 가르쳐주는 사람이지, 외국인을 만나 나눌 대화의 대본을 만들어 주는 사람이 아니다. 또 언어를 배운 후에는 언어를 사용해봐야 잊어버리지 않는다. 마찬가지로 이 책을 통해 C 언어를 사용하는 방법을 익힌 뒤, 독자 여러분도 그 언어를 사용해 각자 만들어보고 싶었던 프로그램을 한 번쯤 꼭 작성해보기 바란다.

■ 예제 파일 및 책에서 사용한 기호의 의미

이 책에 포함된 소스코드는 다음 주소를 통해 다운받을 수 있다.

- http://www.hanbit.co.kr/exam/2715

이 책에 사용한 기호는 다음과 같은 의미로 사용하였다.

 표시(error) : 컴파일 시 에러가 발생하는 경우

 표시(warning) : 컴파일 시 에러가 발생하진 않지만, 바람직하지 않은 경우

한빛 eBook 리얼타임

한빛 eBook 리얼타임은 IT 개발자를 위한 eBook입니다.

요즘 IT 업계에는 하루가 멀다 하고 수많은 기술이 나타나고 사라져 갑니다. 인터넷을 아무리 뒤져도 조금이나마 정리된 정보를 찾는 것도 쉽지 않습니다. 또한 잘 정리되어 책으로 나오기까지는 오랜 시간이 걸립니다. 어떻게 하면 조금이라도 더 유용한 정보를 빠르게 얻을 수 있을까요? 어떻게 하면 남보다 조금 더 빨리 경험하고 습득한 지식을 공유하고 발전시켜 나갈 수 있을까요? 세상에는 수많은 종이책이 있습니다. 그리고 그 종이책을 그대로 옮긴 전자책도 많습니다. 전자책에는 전자책에 적합한 콘텐츠와 전자책의 특성을 살린 형식이 있다고 생각합니다.

한빛이 지금 생각하고 추구하는, 개발자를 위한 리얼타임 전자책은 이렇습니다.

1. **eBook Only - 빠르게 변화하는 IT 기술에 대해 핵심적인 정보를 신속하게 제공합니다.**

   500페이지 가까운 분량의 잘 정리된 도서(종이책)가 아니라, 핵심적인 내용을 빠르게 전달하기 위해 조금은 거칠지만 100페이지 내외의 전자책 전용으로 개발한 서비스입니다. 독자에게는 새로운 정보를 빨리 얻을 수 있는 기회가 되고, 자신이 먼저 경험한 지식과 정보를 책으로 펴내고 싶지만 너무 바빠서 엄두를 못 내는 선배, 전문가, 고수 분에게는 보다 쉽게 집필할 수 있는 기회가 될 수 있으리라 생각합니다. 또한 새로운 정보와 지식을 빠르게 전달하기 위해 O'Reilly의 전자책 번역 서비스도 하고 있습니다.

2. **무료로 업데이트되는, 전자책 전용 서비스입니다.**

   종이책으로는 기술의 변화 속도를 따라잡기가 쉽지 않습니다. 책이 일정 분량 이상으로 집필되고 정리되어 나오는 동안 기술은 이미 변해 있습니다. 전자책으로 출간된 이후에도 버전 업을 통해 중요한 기술적 변화가 있거나 저자(역자)와 독자가 소통하면서 보완하여 발전된 노하우가 정리되면 구매하신 분께 무료로 업데이트해 드립니다.

3. 독자의 편의를 위하여 DRM-Free로 제공합니다.

구매한 전자책을 다양한 IT 기기에서 자유롭게 활용할 수 있도록 DRM-Free PDF 포맷으로 제공합니다. 이는 독자 여러분과 한빛이 생각하고 추구하는 전자책을 만들어 나가기 위해 독자 여러분이 언제 어디서 어떤 기기를 사용하더라도 편리하게 전자책을 볼 수 있도록 하기 위함입니다.

4. 전자책 환경을 고려한 최적의 형태와 디자인에 담고자 노력했습니다.

종이책을 그대로 옮겨 놓아 가독성이 떨어지고 읽기 힘든 전자책이 아니라, 전자책의 환경에 가능한 한 최적화하여 쾌적한 경험을 드리고자 합니다. 링크 등의 기능을 적극적으로 이용할 수 있음은 물론이고 글자 크기나 행간, 여백 등을 전자책에 가장 최적화된 형태로 새롭게 디자인하였습니다.

앞으로도 독자 여러분의 충고에 귀 기울이며 지속해서 발전시켜 나가도록 하겠습니다.

차례

6 | 함수

6장의 목표

- 함수의 의미와 구조를 이해한다.
- 함수를 호출하고 그 반환값을 사용하는 코드를 작성해본다.
- 코드상에서 함수를 분리해야 하는 경우와 모듈화의 의미를 이해한다.
- 재귀함수의 구조를 이해하고 필요한 상황에 맞게 작성해본다.
- 프로그램, 혹은 함수 내에서 유효한 변수를 선언하는 방법을 이해한다.

6장의 구성

작은 부품이 모여 하나의 큰 장치를 완성하듯, 여러 개의 함수가 모여 하나의 프로그램을 완성한다. 6장에서는 프로그램의 부품이라 할 수 있는 함수의 형태와 기능을 알아보고 자유롭게 사용할 수 있도록 연습해본다.

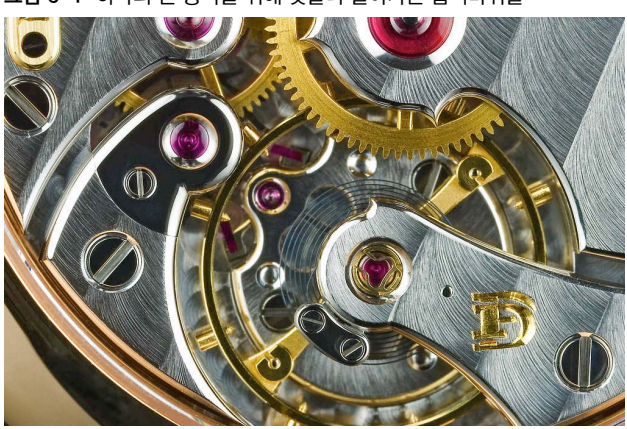
6.1 함수란 무엇인가?

C 언어의 기본 문법을 이해하기 위해 여러 장에 걸쳐 하나의 함수를 사용했다(『BACK TO THE BASIC, 뿌리부터 이해하는 C 언어(Vol.1)』 참조). 그것은 바로 main 함수로, 컴파일러의 옵션에 따라 새로운 프로젝트를 생성하면 자동으로 작성되는 함수다. 컴파일러는 왜 main 함수를 자동으로 생성해줄까? 그리고 함수란 무엇일까?

```
void main( void )
{
     printf("hello world!");
}
```

하나의 장치를 구성하는 여러 개의 톱니바퀴를 떠올려보자. 그 장치는 맞물려진 여러 개의 톱니바퀴가 움직여 동작하는데, 이를 위해서는 반드시 최초의 움직임을 시작하는 톱니바퀴가 있어야 한다. 그 톱니가 움직일 때야 비로소 맞물려 연결된 모든 톱니바퀴가 동작하게 되는 것이다. C 프로그램 내에서, 이처럼 최초로 움직이는 톱니바퀴가 바로 main 함수다.

그림 6-1 하나의 큰 동작을 위해 맞물려 돌아가는 톱니바퀴들

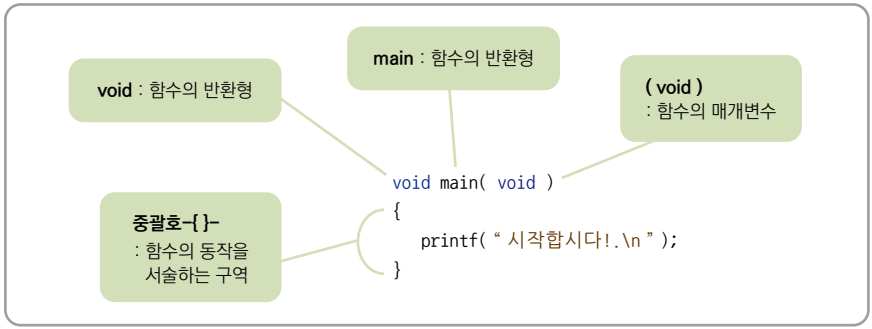


장치가 동작하기 위해 움직임을 시작하는 톱니바퀴 하나가 꼭 필요하듯 C 언어에서 하나의 프로그램은 반드시 하나의 main 함수가 있어야 한다. 프로그램을 구동한다는 것은 곧 그 프로그램의 main 함수를 동작시킨다는 의미다. 컴퓨터를 켜고 C로 만들어진 어떤 프로그램을 더블클릭하면, 컴퓨터는 이 프로그램 내의 main 함수를 찾아 함수 내에 있는 모든 코드를 해석하여 동작시킨 후 프로그램을 종료한다. 즉, 프로그램의 시작과 끝은 main 함수의 시작과 끝이다.

지금까지 예제에서는 main 함수 하나에 모든 코드를 작성했으나 코드가 길고 복잡해지면 main 함수 하나로 프로그램을 완성하는 데 점점 더 어려움을 느끼게 될 것이다. 이런 문제를 해결하기 위해 C 언어에서는 main에서 동작해야 할 코드를 분리하여 다른 함수로 작성하는 방법을 제공한다. 개발자는 필요한 경우, 제공되는 형식에 따라 함수들을 새로 정의하고 호출하여 사용할 수 있다.

다시 말하면 여러 개의 함수로 프로그램을 구성할 수 있는데, 이 함수들은 main 함수를 기점으로 동작한다. main 함수는 필요에 따라 다른 함수를 호출해 동작시키며 그 함수들은 또 다른 함수를 동작시키기도 한다. 이러한 일련의 과정을 통해 프로그램이라는 큰 장치가 동작하는 것이다. 이 때문에 많은 컴파일러에서는 프로젝트 생성 시 main 함수를 자동으로 생성해준다. 만약 main 함수가 생성되지 않았다면 여러분이 직접 작성해야 한다.

main 함수를 포함하여, 프로그램에 사용되는 함수는 모두 동일한 구조로 되어 있다. 그럼, 지금부터 여러 번 사용하면서도 뭔지 몰랐던 main 함수를 살펴봄으로써 함수의 기본 구조를 확인해보자. 기본 구조를 익힌 후에는 새로운 함수를 만들고 호출해볼 것이다.



6.1.1 함수의 반환형

함수는 프로그램 내에서 작동하는 여러 동작 중 일부 동작을 전담한다. 일상생활에서 다른 누군가에게 어떤 동작을 하도록 요청하는 경우를 생각해보자. 그 요청은 첫째, 단순히 정말 다른 사람에게 '어떤 동작만'을 요청하는 경우일 수 있다. 예를 들어 "조용히 해주세요", "오른손을 들어주세요", "눈을 깜박여보세요"와 같이 말 그대로 그 사람에게 어떤 동작만을 요청하는 경우다. 두 번째로 어떤 동작을 통한 '결과물'을 요청하는 경우가 있다. "거기 있는 커피잔 좀 집어주세요" 혹은 "이거 쓰고 나면 회비가 얼마 남지?"와 같은 요청은 상대방에게서 커피잔을 건네받거나 남은 회비의 액수를 계산한 결과를 전달받기 위한 요청이다.

함수 역시 어떤 동작을 전담할 때, 단지 함수 내에서 동작이 끝나는 것 자체로 역할이 완료되거나 어떤 동작을 모두 마친 후 함수를 호출한 곳으로 결과값을 반환하기도 한다.

호출된 함수가 단지 함수 내에 정의된 동작만 완료하는 경우라면 이 함수는 반환할 값이 없다. 이런 경우 반환형으로 '하나도 없다'는 의미인 'void'를 선언하거나 아무것도 적지 않는다. 즉, 앞에서 보았던 main 함수는 함수가 종료될 때 아무것도 반환하지 않으며 다음과 같이 사용해도 정상적으로 동작한다.

```
main( void ) //함수의 이름 앞에 아무런 반환형도 적지 않았다.
{
    printf("시작합시다!.\n");
}
```

어떤 값을 반환하는 함수라면 함수를 정의할 때 반드시 반환형을 표기해야 한다. 예를 들어 현재로부터 5시간 후의 시각을 반환해주는 'give_oclock_after_5h' 함수를 만든다고 생각해보자. 반환되는 값은 시각, 즉 정수값이 될 것이므로 이 함수의 반환형을 'int'로 하는 것이 적절하다. 따라서 다음과 같이 함수를 선언한다.

```
int give_oclock_after_5h( … )
{
    …
}
```

이번에는 현재 전교에서 성적이 제일 좋은 학생의 이름을 반환해주는 'top_score_student' 함수를 만든다고 생각해보자. 이러한 경우 문자열이 반환되어야 하므로 반환형은 'char*'로 하는 것이 적절하다. 따라서 다음과 같이 선언한다.

```
char* top_score_student( … )
{
    …
}
```

특정 형태의 값을 반환하겠노라 선언했다면 실제 함수 내에서는 그 값을 어떻게 반환할까? 값의 반환 문법은 간단히 '반환'이라는 사전적 의미를 가진 영단어 'return'을 사용한다. return이라는 단어 뒤에 오는 값은 함수가 호출된 곳으로 반

환되는 값이다. 함수의 정의에서 선언된 형태와 반환되는 값의 형태는 반드시 일치해야 한다. 즉, 앞에서 살펴본 'give_oclock_after_5h' 함수는 다음과 같이 코딩할 수 있다. int를 반환하겠노라 선언했으며, return 뒤에 int형 변수를 반환한다.

```
int give_oclock_after_5h( … )
{
    int resultTime;
    …
    return resultTime;
}
```

이와 같은 return문은 값을 반환하면서 함수를 종료시킨다.

Special Tip 6-1 return과 break의 차이

조건문 중 switch-case에 대해 이야기하면서 break를 잠깐 언급했다. 각 case문의 마지막에 break를 작성하면, break문 아래의 case문을 검사하지 않고 switch문을 빠져나가게 된다는 내용이었다.

이 break 대신 사용할 수 있는 문법이 return이다. return을 사용하는 경우에도 해당 switch문을 빠져나간다. 하지만 break는 해당 switch문만을 빠져나가 다음 코드를 계속해서 동작시키지만, return의 경우 switch문뿐만 아니라 함수를 빠져나가서 해당 함수 내의 동작을 더 이상 진행시키지 않는다는 차이가 있다.

예를 들어 다음 코드에서 chossedNum이 1인 경우 switch문 아래에 있는 printf가 실행되지만, 그 값이 2인 경우 이 위치에서 함수를 종료시키고 호출한 곳으로 복귀하므로 아래쪽의 printf문은 실행되지 않는다.

```
switch(choosedNum)
{
```

```
    case 1:
        break;
    case 2:
        return;
    default:
        break;
}

printf("switch-case end\n");
```

return이란 해당 위치에서 함수를 종료시킨다는 점을 꼭 기억하길 바라며 실행되지 않을 코드를 return문 아래 쪽에 배치하는 실수를 하지 않도록 하자.

표 6-1 반환형의 정의

	`void sumNum(int a, int b)`
	`{`
	`  ...`
	`  return a + b;`
		반환형으로 void를 정의하고 있는데, 반환값은 int값이다. 즉, 잘못된 함수 정의다.

	`sumNum(int a, int b)`
	`{`
	`  ...`
	`  return ;`
		반환형의 표시가 없다는 것은 void와 동일하다. return 시 아무것도 반환하지 않으므로 정상적인 함수 정의다.

	`int sumNum(int a, int b)`
	`{`
	`  int sum = a + b;`
	`}`
		int를 반환하는 것으로 선언한 후 정작 함수 내에서는 아무런 반환값이 없다. 즉, 잘못된 함수 정의다.

	`int sumNum(int a, int b)`
	`{`
	`  char* resultString = "Calculate  end";`
	`  int sum = a + b;`
	`  return resultString;`
	`}`
		int를 반환하는 것으로 선언한 후 return 시에는 문자열을 반환하고 있다. 즉, 잘못된 함수 정의다.

```
int sumNum(int a, int b)
{
   return a + b;
}
```

int를 반환하는 것으로 선언한 후 int값을 반환하고 있다. 정상적 함수 정의다.

```
float sumNum(int a, int b)
{
   return a + b;
}
```

int와 float 간에는 숫자라는 공통점이 있어 괜찮을 것 같지만, 이 함수를 실행하면 에러가 발생한다. float 형식의 반환형을 선언했다면 return값은 float형이어야 한다.

```
int sumNum(int a, int b)
{
   int sum = a + b;
}
void main()
{
   char* result;
   result = sumNum(1, 3);
   ...
```

int형의 반환형을 갖는 함수를 호출하면서 함수의 반환값을 char*형의 변수에 넣으려고 시도하고 있다. 이러한 코드는 실행 시 에러가 발생한다.

6.1.2 함수의 이름

모든 프로그램에서 반드시 가져야만 하는 함수는 main이라는 이름이 정해져 있다. 프로그램이 구동되면 컴퓨터에서는 먼저 이름이 main인 함수를 찾아와 그 내용을 한 줄 한 줄 동작시킨다. 이외에 우리가 직접 만들 함수들은 몇몇 규칙만 지킨다면 정해진 것 없이 원하는 이름을 붙일 수 있다. 함수 이름을 만드는 규칙은 변수 이름을 만들 때의 규칙과 거의 비슷하다. 그래서 상세 설명 없이 간단히 다시 한 번 언급하는 것으로 대신한다. 무엇보다 가독성에 관한 이슈는 변수명과 함수명, 혹은 모든 코드 작성에도 적용되는 변함 없이 중요한 부분이라는 것을 늘 기억하길 바란다.

① 띄어쓰기는 허용하지 않는다.

: go to first( ) 라는 함수명은 사용할 수 없다. gotofirst( )라고 선언해야 한다.

② 대문자와 소문자를 구별한다.

: 컴파일러는 함수 이름을 확인할 때 대소문자를 구별한다. 가령, goToFirstTap( )과 gotofirsttap( )은 다른 함수다. goToFirstTap( )이라는 함수를 선언하고서 gotofirsttap( )이라는 함수를 불러봐야 컴파일러는 오류만을 보여줄 뿐이다.

③ 읽기 편하게 만든다.

: 띄어쓰기가 허용되지 않는다고 해서 함수 이름을 문장형으로 쓸 수 없는 것은 아니다. 변수에서와 마찬가지로 대소문자와 '_'를 잘 활용하면 가독성이 있는 문장형의 함수명을 만들기에 충분하다.

④ 어떤 동작을 위한 함수인지 함수명에 명확히 표현한다.

: gotofirst라는 이름을 접한 사람은 적어도 이 함수가 무엇인가의 처음으로 돌아가는 함수라는 것을 명확히 이해할 수 있을 것이다.

⑤ 알파벳 문자열, 혹은 '_'로 시작되어야 한다.

: 변수와 마찬가지다. 컴파일러는 알파벳 문자열 혹은 '_'로 시작하는 이름을 허용한다.

⑥ 예약어를 함수 이름으로 사용할 수 없다.

: 예약어에는 여러 가지가 포함될 수 있는데, if, else, goto, int, float 등 C 언어에서 이미 어떤 의미를 위해 규정하고 있는 단어들이 모두 포함된다. 또한 기본적으로 사람들이 많이 사용할 것 같은 동작을 미리 만들어놓은 기본 함수들의 이름도 이에 포함된다.

이외에 어떤 규제가 있을까? 언급된 규칙을 모두 지킨 함수 이름에서 컴파일 에러 등의 문제가 발생할 확률은 극히 낮을 것으로 예상하지만 그 외에 알고 있어야 할 규칙이 있다면 여러분이 직접 체험하여 찾아보길 바란다. 최고의 지식 습득 방법은 체험이다.

6.1.3 함수의 매개변수

때로 함수는 동작을 위해 어떤 값을 전달받을 필요가 있다. 규모가 큰 식당의 주방을 머릿속에 떠올려보자. 이런 식당에는 대개 메인 셰프와 셰프를 돕는 보조 주방장들 및 잔업 담당자들이 있다. 하나의 요리가 완성되기까지 메인 셰프는 주위의 보조 주방장 및 잔업 담당자들을 쉴 새 없이 호출하여 요리를 진행시킨다. 이 과정에서 셰프는 단지 그들에게 동작만 지시하는 것이 아니라, 부주방장에게 요리접시를 넘겨 장식을 시키거나 설거지 담당에게 사용한 그릇들을 넘기는 등, 처리해야 할 개체를 지시와 함께 전달한다.

그림 6-2 협력하여 요리하는 것처럼 함수의 매개변수는 적절히 전달해야 한다.

메인 셰프가 요리를 진행하는 과정에서 설거지 담당에게 장식만 남은 요리를 전달하거나, 부주방장에게 설거지거리를 넘겨버린다면 이 주방에는 큰 혼란이 발생할 것이다. 각 담당자가 전달받을 값이 정해져 있듯, 프로그램에서도 이처럼 각 동작을 담당하는 함수가 전달받을 값을 정의해두는데, 그 방법이 바로 '매개변수의 선언'이다.

함수를 정의할 때 이름 뒤에 괄호를 쓰고, 이 괄호 안에 해당 함수에 전달될 매개변수들을 선언한다. main 함수의 경우 프로그램 시작 시점에서 전달받을 내용이 있

는 경우가 드물기 때문에 주로 (void)와 같이 매개변수가 선언되곤 한다. 반환형에서와 마찬가지로 매개변수에서 void라는 것 또한 매개변수가 없다는 의미로 사용되며, void라는 선언을 하는 것과 아무것도 적지 않는 것은 동일한 의미를 지닌다. 따라서 앞서 선언된 main 함수는 다시 한 번 다음과 같이 줄일 수 있다.

```
main()
{
    printf("시작합시다!.\n");
}
```

함수에 값을 전달할 때 쓰인다는 특징을 제외하면 매개변수는 일반변수와 크게 다르지 않다. 일반변수 선언과 동일하게 변수형의 선언 후 이름을 선언하며, 각 매개변수는 ','로 구분한다. 이렇게 선언된 매개변수는 함수 내에서 일반변수와 동일하게 사용된다.

이렇게 매개변수가 정해져 있는 함수에 값을 전달하는 것을 '인자를 전달한다'라고도 표현한다. 매개변수와 인자라는 단어는 종종 혼용되어 사용되지만, 정확히 하자면 함수 생성 시 선언되는 전달 값의 형태를 매개변수라고 표현하며 그 함수의 매개변수에 전달하는 값을 인자라고 표현하는 것이 맞다. 즉, 매개변수는 말 그대로 '변수'며, 인자란 그 변수로 넘겨지는 '값'을 뜻한다. 물론 프로그래밍을 하면서 일일이 이런 용어를 구별할 필요는 없지만, 둘의 차이를 한 번쯤은 인지하고 넘어가기 바란다.

함수가 동작하기 위해 전달받을 값의 개수는 정해져 있지 않다. 개발자는 필요한 만큼 매개변수를 선언할 수 있다. 가령, 세 개의 숫자를 전달받아 그 합을 반환하는 함수라면 세 개의 숫자를 매개변수로 받는다고 선언해야 하므로 다음과 유사한 형태의 함수가 선언될 것이다.

```
int sumup_from_inserted_numbers(int firstNum, int secondNum, int thirdNum)
```

Special Tip 6-2 매개변수가 있는 main 함수

우리가 만든 main 함수는 반환형도 void였으며 매개변수형도 void였다. 아무것도 전달받지 않고 아무것도 반환하지 않겠다는 뜻이다. 하지만 때에 따라서는 main 함수도 어떤 값을 전달받을 수 있는데, 이 경우 main 함수에서 받을 수 있는 인자값의 형태는 이미 다음과 같이 정해져 있다.

```
void(or int) main(int argc, char** argv, char** env )
```

이 매개변수들은 뒤에서부터 하나씩 삭제가 가능하다. 즉 main(int argc, char** argv), main(int argc) 혹은 main( )과 같은 선언이 가능하다. 하지만 마음대로 매개변수를 변경하거나 지정할 수는 없다. main 함수는 이와 같이 이름도 매개변수도 컴퓨터와 이미 약속된 특수한 함수다.

그렇다면 main 함수에 인자를 어떻게 전달할까? main 함수는 프로그램의 시작 시 동작하는 함수다. 따라서 프로그램 시작 시 인자를 전달해줘야 한다. 그런데 일반적으로 프로그램은 아이콘을 더블클릭함으로써 시작하며, 이때 사용자는 따로 값을 전달할 방법이 없다.

따라서 명령 프롬프트를 사용해서 프로그램을 시작하는 경우에만 인자를 main 함수에 전달할 수 있다. 1권의 1장에서 명령 프롬프트를 사용했던 것을 기억해보자. 프로그램을 실행하기 위해서 우리는 프로그램의 파일 이름을 입력하고 [Enter] 키를 입력했다. 이때 파일 이름 뒤에 띄어쓰기한 후 원하는 값을 입력하면, 이 값은 문자열 배열의 형태로 main 함수에 전달된다. 여러 개의 인자를 입력하고 싶다면 '파일이름 인자 인자 인자' 형식으로 입력 인자 사이에 띄어쓰기를 넣는다.

main 함수의 매개변수인 'int argc'와 'char** argv'에는 인자의 개수와 인자를 배열로 연결한 문자열 배열이 전달된다. 사용자가 인자를 하나도 넣지 않은 경우에도 하나의 인자는 무조건 main 함수로 넘어오는데, 그것은 '프로그램명'이다. 그러므로 전달된 배열의 두 번째 값부터 사용자가 입력한 값들이 들어오게 된다. 다음의 프로그램을 통해 이와 같은 동작을 확인해보자.

예제 6-1 main 함수에서 전달받은 인자의 출력 **(예제 파일 : ex06-01.c)**

```
#include <stdio.h>

main(int argc, char** argv)
{
  int i = 0;
```

```
    //전달된 인자의 개수를 출력
    printf("Num of arguments : %d\n", argc);

    //전달된 문자열 배열을 차례대로 출력
    for(i; i < argc; ++i)
    {
        printf("%d : %s\n", i, argv[i]);
    }

    return 0;
}
```

프로그램을 실행하여 인자를 전달하지 않은 경우와 전달한 경우의 결과를 각각 확인해보자. 인자가 어떻게 전달되는지 이해할 수 있을 것이다.

```
C:\User\andro\Documents]\Visual Stduio 2013\Projects\Ex_Main_Parameter\Debug
>Ex_Main_Parameter.exe
Num of arguments : 1
0 : Ex_Main_Parameter.exe
C:\User\andro\Documents]\Visual Stduio 2013\Projects\Ex_Main_Parameter\Debug
>Ex_Main_Parameter.exe ManU MacCity Chelsea Liverpool Arsennal
Num of arguments : 1
1 : ManU
2 : MacCity
3 : Chelsea
4 : Liverpool
5 : Arsennal
```

마지막 매개변수인 env에는 운영체제의 환경변수가 넘어오지만, 환경변수를 확인하는 다른 방법도 많으며, 프로그램 내에서 이를 잘 사용하지 않는다.

Special Tip 6-3 매개변수의 유효성 검사

함수는 매개변수로 전달받을 값의 형태를 미리 정의하고 있지만, 그럼에도 불구하고 해당 형에 맞지 않는 값이 넘어가는 경우가 더러 있다. 코딩을 잘못한 결과이기도 하고 때로는 뜻하지 않은 예외 상황에 의한 것이기도 하다. 이러한 경우 에러가 발행하지 않도록, 함수에서는 매개변수의 유효성을 검사해야 한다.

예를 들어 전달받은 값을 문자열로 간주하고 다른 문자열과 비교하는 다음 함수를 보자.

```
void studentSearch(char* studentName)
{
  char* LisedtName = "Benedict";    //비교대상이 될 문자열
  int searchrResult = 0;            //비교 결과를 담을 변수

  //전달된 변수를 ListedName에 담긴 문자열과 비교
  searchrResult = strcmp(LisedtName, studentName);

  //같은 문자열이라면 이를 출력하고 아니라면 다른 사람이란 것을 출력
  if(searchrResult == 0)
     printf("Matched : %s", LisedtName);
  else
     printf("different person");
}
```

이 함수를 부르는 곳에서 "studentSearch("Benedict");"와 같이 정상적으로 문자열을 전달해주면 이 함수는 정상적으로 동작하지만, "studentSearch(NULL);"와 같이 잘못된 인자를 넘겨주면 이 프로그램은 비정상 종료된다. 비정상 종료를 막기 위해서 필요한 것이 바로 다음과 같은 유효성 검사다.

예제 6-2 변수의 유효성 검사 (예제 파일 : ex06-02.c)

```
void studentSearch(char* studentName)
{
```

```
    char* LisedtName = "Benedict";    //비교대상이 될 문자열
    int searchrResult = 0;            //비교 결과를 담을 변수

    if(studentName == NULL) //전달된 값이 NULL이라면 return으로 함수를 종료함
        return;

    char* LisedtName = "Benedict";
    …

}
```

위와 같은 매개변수의 유효성 검사는 잘못된 값이 전달된 경우에도 프로그램이 비정상 종료되지 않고 진행될 수 있도록 돕는다. 필요에 따라서는 return값에 의미를 부여해서 return 값이 1이라면 정상 동작, 0이라면 비정상 동작으로 분류하는 식의 처리도 가능하다.

6.1.4 중괄호: 함수의 영역

1권의 "3장 조건문"에서 if 조건문 다음에 오는 중괄호는 if 조건문이 미치는 영역을 한정한다고 설명했다. 마찬가지로 함수 선언 시 괄호 다음에 오는 중괄호({ })는 그 함수의 영역을 나타낸다. 다만 if 조건문 다음에는 중괄호가 올 수도 있고 오지 않을 수도 있었으나, 함수를 정의할 때는 반드시 중괄호를 통해 함수 본체가 있음을 나타내야 한다. 중괄호 영역에서는 함수가 동작할 내용을 서술한다.

Special Tip 6-4 여러 개의 함수 정의

상용 프로그램의 코드는 그 양이 매우 많으며, 그 안에는 수십 수백 개의 함수가 있다. 이렇게 여러 개의 함수를 만들다 보면 역할이 비슷한 함수를 만드는 일도 종종 발생하며, 비슷한 역할에 따라 비슷한 이름의 함수를 만드는 일도 자주 발생한다. C 언어에서 허용하는 함수명의 유사성을 확인하기 위해 다음의 코드를 빌드해보자.

예제 6-3 여러 이름의 함수 선언 (예제 파일 : ex06-03.c)

```c
#include <stdio.h>

int sumFunction(int a, int b)            // 함수 1
{
}

int sumfunction(int a, int b)            // 함수 2
{
}

void sumFunction(int a, int b)           // 함수 3
{
}

int sumFunction(int a, int b, int c)     // 함수 4
{
}

main()
{
  //어디에서 에러가 발생할까?
}
```

위 코드에서 main 함수 위에 정의된 함수 4개는 모두 비슷비슷한 이름으로 되어 있다. 하지만 에러가 발생하는 것은 3번 함수뿐이다.

먼저, 1번 함수는 이전에 비교되거나 중복되는 함수가 없으므로 당연히 에러가 발생하지 않는다. 2번 함수는 1번 함수와 이름이 매우 비슷하지만, 중간의 철자 'f'가 대문자인 1번과 달리 소문자로 되어 있다. 이처럼 대소문자를 달리하는 경우, 이 두 함수는 서로 다른 함수로 간주되기 때문에 컴파일 시 에러가 발생하지 않는다.

함수 4번의 경우 1번 함수와 다른 모든 것이 동일하지만 매개변수를 하나 더 가지고 있다는 점이 다르다. 매개변수의 개수가 다른 경우 서로 다른 함수로 인식되어 컴파일에 문제가 되지 않는다.

문제가 되는 것은 3번이다. 1번과 3번 함수는 반환형만 다를 뿐, 다른 모든 것이 같다. 이러한 경우, 컴파일러는 동일한 함수가 재정의된 것으로 인식하여 에러가 발생한다.

6.1.5 함수 호출하기

main 이외의 함수는 호출될 때에만 동작한다. 함수를 동작시키는 것을 '호출하다', '부르다'라고 표현하는 것은 일상에서 사용하는 표현이 그대로 차용된 때문이다. 누군가를 부르는 동작은 간단하게 그 사람의 이름 혹은 별명, 직급 등을 외치는 행위다. 드라마에서는 종종 부잣집 사모님이 목청 높여 '김 비서!!', '아줌마!', '김 기사!!' 등을 부르는(=호출하는) 장면을 볼 수 있듯, 함수를 호출할 때도 역시 이와 마찬가지로 함수의 이름을 사용한다.

매개변수가 없는 함수라면 단지 '함수이름( )'의 형식으로 아무 내용이 없는 괄호를 덧붙여주며, 매개변수가 있는 함수라면 괄호 내에 필요한 인자를 함께 전달해준다. 앞에서 매개변수를 3개 전달하도록 선언한 함수를 상기해보자.

```
int sumup_from_inserted_numbers( int firstNum, int secondNum, int thirdNum )
```

이와 같이 매개변수가 있는 함수의 호출은 다음과 같이 여러 형식으로 이루어질 수 있다.

① 매개변수 형에 맞는 상수 전달

```
sumup_from_inserted_numbers(1, 2, 3);
```

② 매개변수 형에 맞는 변수 전달

```
int firstN = 0, secondN = 0, thirdN = 0;
...
sumup_from_inserted_numbers(firstN, secondN, thirdN);
```

③ 매개변수 형에 맞는 결과값을 갖는 수식, 함수 전달

```
int secondInput( ) { return 3; };
int firstN = 0, secondN = 0, thirdN = 0;
...
sumup_from_inserted_numbers( 1*3, secondInput( ), thirdN + 3 );
```

이제 우리는 함수의 구조를 알게 됐고 이에 맞추어 작성된 함수를 필요에 따라 호출할 수 있다. 표 6-2에서 이러한 함수 호출의 다양한 예를 제시해두었으니, 표를 통해 언제 에러가 발생하는지 잘 살펴보기 바란다.

표 6-2 함수의 호출

```
#include <stdio.h>

int sumFunction(int a, int b)
{
   return a + b;
}

main()
{
   int sumNumber = sumFunction(1, 3);
   printf("1 + 3 = %d", sumNumber);
}
```

- 정상 호출

```
#include <stdio.h>

int sumFunction(int a, int b)
{
   return a + b;
}

main()
{
    int sumNumber = sumfunction(1, 3);
    printf("1 + 3 = %d", sumNumber);
}
```

- 컴파일 에러 발생
- 함수 이름 에러
- 대소문자를 잘못 표기하여 호출함

```c
#include <stdio.h>

int sumFunction(int a, int b)
{
   return a + b;
}

main()
{
   int sumNumber = sumFunction();
   printf("1 + 3 = %d", sumNumber);
}
```

- 컴파일 에러 발생
- 매개변수가 모자람

```c
#include <stdio.h>

int sumFunction(int a, int b)
{
   return a + b;
}

main()
{
   int sumNumber = sumFunction(1, 3, 4);
   printf("1 + 3 = %d", sumNumber);
}
```

- 매개변수가 넘침
- 컴파일 에러는 발생하지 않으나 잘못된 호출임. 코드에 명확성을 떨어트리는 코드임

```c
#include <stdio.h>

int sumFunction(int a, int b)
{
   return a + b;
}

main()
{
   int sumNumber = sumFunction(1);
   printf("1 + 3 = %d", sumNumber);
}
```

- 컴파일 에러 발생
- 매개변수가 모자람

```c
#include <stdio.h>

int sumFunction(int a, int b)
{
  return a + b;
}

main()
{
  int firstNum = 1;
  int sumNumber = sumFunction(firstNum, 3);
  printf("1 + 3 = %d", sumNumber);
}
```

• 정상 호출

```c
#include <stdio.h>

int sumFunction(int a, int b)
{
  return a + b;
}

main()
{
  int firstNum = 1;
  int sumNumber = sumFunction(firstNum + 4, 3);
  printf("1 + 3 = %d", sumNumber);
}
```

• 정상 호출

6.2 함수, 모듈화

main 함수 이외의 다른 함수는 왜 필요할까? 왜 굳이 여러 개의 함수를 만들어야 할까? 사실 코드 작성자가 원한다면 main 함수 하나로 프로그램을 작성할 수도 있다. 앞의 예제에서 함수를 분리하여 작성한 프로그램들도 main 함수 내에서 작성 가능하다. 그렇다면 함수 생성과 사용의 장점은 무엇일까?