

Hanbit eBook

Realtime 97



모던 C++

C++14의 핵심 기능을 중심으로

최흥배 지음

모던 C++

C++14의 핵심 기능을 중심으로

최종매 지음

 **한빛미디어**
Hanbit Media, Inc.

모던 C++ C++14의 핵심 기능을 중심으로

초판발행 2015년 3월 31일

지은이 최흥배 / **펴낸이** 김태현

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 978-89-6848-746-0 15000 / **정가** 6,000원

총괄 배용석 / **책임편집** 김창수 / **기획·편집** 정지연

디자인 표지/내지 여동일, 조판 최송실

마케팅 박상용 / **영업** 김형진, 김진불, 조유미

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

Copyright © 2015 최흥배 & HANBIT Media, Inc.

이 책의 저작권은 최흥배와 한빛미디어(주)에 있습니다.

저작권법에 의해 보호를 받는 저작물이므로 무단 복제 및 무단 전재를 금합니다.

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.

지은이_ **최흥배**

2003년부터 PC 보드 게임에서 MMORPG, 모바일 게임까지 아우르는 다양한 온라인 게임 서버 프로그램을 만들어온 게임 개발자다. 프로그래밍 언어는 C++과 C#을 주로 사용하며, 요즘은 C++11/14 프로그래밍과 심도 있는 .NET 기술, 유명 백엔드 오픈소스 라이브러리와 프로그램, Go 프로그래밍 언어를 주로 공부하고 있다.

기술과 개발 경험을 여러 사람과 나누는 것을 좋아하여 게임 개발자 커뮤니티와 세미나 강연을 통해 다른 프로그래머와 활발히 교류하고 있다. 웹이 대중화되기 전부터 프로그래밍 공부를 해 와서 그런지 여전히 새로운 기술을 배울 때는 책을 선호하여 지금도 매달 새로운 프로그래밍 관련 책을 읽으며 연구하고 있다.

현재 T3엔터테인먼트(<http://www.t3.co.kr>) 모바일 1팀에서 모바일 게임 서버를 개발 중이다.

블로그 : <http://jacking.tistory.com/>

트위터 : [@jacking75](#)

C++는 Java나 C#보다 낫은 느낌이 들지만 C++11을 기점으로 빠르게 진화 중이다. 2014년 말 새 표준으로 C++14가 나왔고, 현재는 프로그래머들의 요구에 빠르게 대응하기 위해서 큰 변화를 담은 C++1y 버전을 작업 중이다. 따라서 이전에 배운 것만 생각하고 새로 바뀐 C++를 공부하지 않으면 제대로 된 C++ 프로그래밍을 할 수 없다.

대표적 C++ 컴파일러인 Visual C++는 아직 C++11/14의 구현이 부족하지만 올해 나올 최신 버전에서 대부분 구현될 예정이고, Clang과 GCC는 이미 C++11/14를 모두 구현한 컴파일러가 나왔다. 즉, 공부할 수 있는 환경과 현업에서 사용할 수 있는 환경이 거의 다 갖추어진 상태다.

C++14는 이전 표준인 C++11의 마이너 업데이트로, C++11의 부족한 부분을 보완하였다. C++11을 이미 공부하였다면 C++14의 새로운 기능은 분량도 적고 내용도 어렵지 않으므로 쉽게 공부할 수 있을 것이다. C++14 표준 작업이 완료된 지 얼마 되지 않아 관련 글이 별로 없어서 공부에 어려움을 느꼈던 분들에게 이 책이 많은 도움이 되기를 바란다.

매번 부족한 제 글을 매끄럽게 잘 다듬어주시는 정지연 과장님과 한빛미디어 관계자분들에게 부족하나마 감사의 마음을 전한다.

이 책을 집필하느라 집안일을 많이 도와주지 못했는데, 이를 잘 이해해준 아내 선민에게 고마움을 전한다. 또한, 함께 자주 놀아주진 못한 딸 지유에게 미안하고 지유가 앞으로로도 계속 건강하게 자라기를 바란다.



이 책은 C++11을 알고 있는 독자를 대상으로 한다. 따라서 C++11을 아직 공부하지 않았다면 『Thinking About C++11 STL 프로그래밍』⁰¹(한빛미디어, 2014)과 『BACK TO THE BASIC, C++11 핵심 노트: 핵심 주제 12가지로 배우는 C++11』⁰²(한빛미디어, 2014)을 먼저 읽어보기 바란다.

쉽고 빠르게 이 책의 예제 코드를 실행해 볼 수 있도록 다음 온라인 컴파일러를 사용하였다. 예제 코드는 별도로 제공하지 않으며 이 책에 나온 소스 코드를 복사하여 온라인 컴파일러에서 실행하면 결과를 확인할 수 있다.

- 온라인 컴파일러 : <http://melpon.org/wandbox/>
- Clang 3.6 버전 선택

⁰¹ <http://bit.ly/18O4su1>

⁰² <http://bit.ly/1x7E0qs>

한빛 eBook 리얼타임은 IT 개발자를 위한 eBook입니다.

요즘 IT 업계에는 하루가 멀다 하고 수많은 기술이 나타나고 사라져 갑니다. 인터넷을 아무리 뒤져도 조금이나마 정리된 정보를 찾기도 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 First - 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 기회가 되고, 자신이 먼저 경험한 지식과 정보를 책으로 펴내고 싶지만 너무 바빠서 엄두를 못 내는 선배, 전문가, 고수 분에게는 좀 더 쉽게 집필할 기회가 될 수 있으리라 생각합니다. 또한, 새로운 정보와 지식을 빠르게 전달하기 위해 O'Reilly의 전자책 번역 서비스도 하고 있습니다.

무료로 업데이트되는 전자책 전용 서비스입니다

2 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 위해 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 어려운 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.

지금 보시는 전자책에 소유권한을 표시한 문구가 없거나 타인의 소유권한을 표시한 문구가 있다면 위법하게 사용하고 있을 가능성이 큼니다. 이 경우 저작권법에 따라 불이익을 받으실 수 있습니다.

다양한 기기에 사용할 수 있습니다. 또한, 한빛미디어 사이트에서 구매하신 후에는 횡수에 관계없이 내려받을 수 있습니다.

한빛미디어 전자책은 인쇄, 검색, 복사하여 붙이기가 가능합니다.

전자책은 오타자 교정이나 내용의 수정·보완이 이뤄지면 업데이트 관련 공지를 이메일로 알려 드리며, 구매하신 전자책의 수정본은 무료로 내려받으실 수 있습니다.

이런 특별한 권한은 한빛미디어 사이트에서 구매하신 독자에게만 제공되며, 다른 사람에게 양도나 이전은 허락되지 않습니다.

chapter 1 C++14의 새로운 기능 — 001

- 1.1 함수 반환값 타입 추정과 decltype(auto) — 001
- 1.2 2진수 리터럴 — 008
- 1.3 숫자 구분 — 010
- 1.4 [[deprecated]] 속성 — 011
- 1.5 Aggregate와 멤버 변수 초기자의 조합 — 014
- 1.6 크기 지정 메모리 해제 — 017
- 1.7 범용 람다 캡처 — 018
- 1.8 제네릭 람다 — 022
- 1.9 constexpr — 023
- 1.10 변수 템플릿 — 035
- 1.11 STL — 037

chapter 2 C++11의 기능 좀 더 알아보기 — 047

- 2.1 새로운 문자열 리터럴 — 047
- 2.2 사용자 정의 리터럴 — 051
- 2.3 가변인수 템플릿 — 054

C++14의 새로운 기능

1.1 함수 반환값 타입 추정과 decltype(auto)

C++11의 기능 중 가장 유용한 것은 'auto'다. auto 키워드를 사용하면 변수를 초기화할 때 컴파일러가 초기화 값에 따라 타입을 지정해준다.

[auto 키워드 사용 예]

```
// char*
auto NPCName = "BugKing";

// int
auto Number = 1;

// 포인터
int UserMode = 4;
auto* pUserMode = &UserMode;

// 참조
auto& refUserMode = UserMode;
refUserMode = 5;

// STL 반복자
std::list<MCommand*> m_listCommand;
auto iter = m_listCommand.begin();
```

이 예처럼 auto 키워드를 사용하면 C++ 프로그래밍의 표현이 쉬워진다. C++14에서는 이 auto 키워드를 함수 반환값 타입에도 사용할 수 있다. C++11까지만

하더라도 함수의 반환값은 타입을 반드시 지정해야 했으나 C++14에서는 auto를 사용하면 컴파일러가 유추하여 타입을 지정한다.

```
void Test1()
{
    std::cout << "Test1()" << std::endl;
}

int Test2()
{
    int n1 = 0;
    return n1;
}

float Test3()
{
    float f1 = 0;
    return f1;
}
```

앞의 코드에 C++14의 '함수 반환값 타입 추정'을 사용하면 다음 코드가 된다.

[코드 1-1] 함수 반환값 타입 추정을 사용한 함수 정의

```
#include <iostream>

auto Test1()
{
    std::cout << "Test1()" << std::endl;
}

auto Test2()
{
    int n1 = 10;
    return n1;
}

auto Test3()
{
```

```

        float f1 = 12.3f;
        return f1;
    }

int main()
{
    Test1();

    auto n1 = Test2();
    std::cout << "Test2(): " << n1 << std::endl;

    auto f1 = Test3();
    std::cout << "Test3(): " << f1 << std::endl;

    return 0;
}

```

FinishTest1 ()은 반환값이 없으므로 void, Test2 ()는 int 타입의 변수를 반환하므로 int, Test3 ()는 float 타입의 변수를 반환하므로 float로 컴파일러가 추정해 준다.

실행 결과

```

Test1()
Test2(): 10
Test3(): 12.3

```

1.1.1 재귀 함수에서 함수 반환값 타입 추정 사용

함수 반환값 타입 추정은 재귀 함수에서도 사용할 수 있다.

[코드 1-2] 함수 반환값 타입 추정을 재귀 함수에 사용한 예

```

#include <iostream>

auto RecursiveFunc(int n)
{
    if(n <= 0)

```

```

    {
        return 0;
    }

    return n + RecursiveFunc(n-1);
}

int main()
{
    int n1 = RecursiveFunc(10);
    std::cout << n1 << std::endl;

    return 0;
}

```

실행 결과

55

단, 재귀 함수에서 사용할 때는 주의할 점이 있다. 재귀 함수의 첫 번째 반환에서 어떤 타입을 반환할지 컴파일러에게 알려줘야 한다. 반환 타입을 알려주지 않으면 다음 코드처럼 오류가 발생한다.

```

auto RecursiveFunc(int n)
{
    if(n > 0)
    {
        return n + RecursiveFunc(n-1);
    }

    return 0;
}

```

실행 결과

```

prog.cc: In function 'auto RecursiveFunc(int)':
prog.cc:7:20: error: use of 'auto RecursiveFunc(int)' before deduction of

```

```
'auto'
    return n + RecursiveFunc(n-1);

prog.cc:7:37: error: invalid use of 'auto'
    return n + RecursiveFunc(n-1);
```

1.1.2 템플릿 함수에서 함수 반환값 타입 추정 사용

함수 반환값 타입 추정을 템플릿 함수에서 사용할 때는 명시적 구현과 명시적 전문화에도 반드시 함수 반환값 타입 추정을 사용해야 한다.

```
template < typename T > auto Test( T t ) { return t ; }

// 명시적 구현
extern template auto Test( int ) ; // OK
extern template float Test( float ) ; // 오류. 반환값 타입으로 float 지정

// 명시적 전문화
template < > auto Test( short ) ; // OK
template < > int Test( int ) ; // 오류. 반환값 타입으로 int 지정
```

1.1.3 잘못된 함수 반환값 타입 추정 사용

함수 반환값 타입 추정을 사용한 함수가 내부에서 복수의 방법으로 값을 반환할 때 각 반환값의 타입은 서로 일치해야 한다.

다음 예에서 Test1() 함수는 각각 int 타입을 반환하므로 문제가 없지만, Test2() 함수는 첫 번째 반환식에서는 double 타입, 두 번째 반환식에서는 int 타입을 반환해서 컴파일 오류가 발생한다.

```
auto Test1(int n)
{
    if(n < 10)
```

```

        return 0 ;

    return 100 ;
}

auto Test2(int n)
{
    if(n < 10)
        return 0.0 ;

    return 100; // 오류
}

```

함수 반환값 타입 추정을 사용한 함수를 선언(함수 FuncA)만 한 다음, 다른 함수(함수 FuncB)에서 FuncA 함수를 호출하면 FuncA에서 어떤 타입이 반환되는지 추정할 수 없으므로(FuncA 함수의 정의를 알 수 없어서) 컴파일 오류가 발생한다.

```

auto FuncA();
//auto FuncA() { return 100; } 이 주석을 풀면 컴파일 오류를 해결할 수 있다.

auto FuncB() { return FuncA(); }

int main()
{
    auto n1 = FuncB();

    return 0;
}

```

클래스의 멤버 함수에서도 함수 반환값 추정을 사용할 수 있지만, 가상 함수에는 사용할 수 없다. 가상 함수에서 함수 반환값 추정을 사용하는 것은 기술적으로 크게 어렵지 않지만, 오버라이드 조사나 vtable 레이아웃이 복잡해지기 때문에 현재는 금지하였다(나중에 허용될 수도 있다).

```

class Test
{
public:
    Test() {}
    ~Test() {}

    auto T1() { return 100; }

    virtual auto T2() { return 1000; } // 여기서 컴파일 오류 발생
};

```

초기화 리스트(`std::initializer_list`)에서는 함수 반환값 추정을 사용할 수 없다. 구현상 초기화 리스트는 저장소에 만든 후 참조로 전달하므로 함수 반환값 추정에 사용하는 것은 적절하지 않다(함수 반환 시 지역 변수를 참조로 반환한다는 의미가 되기 때문이다).

```

auto Test1()
{
    return { 7, 77, 777 }; // std::initializer_list 반환은 오류
}

```

1.1.4 decltype(auto)

`decltype(auto)`는 `auto`로 해결할 수 없는 문제를 해결하기 위해 만들어졌으며 함수 반환값 추정과도 연관이 있다. 다음 코드에서 `r1`이 참조 타입이라고 예상하겠지만 `auto` 문법에 따라 `r1`은 `int` 값 타입이 된다.

```

int n1 = 7;
auto& Func() { return n1;}
auto r1 = Func();

```

`r1`이 참조 타입이 되려면 다음과 같이 작성해야 한다.

```
int n1 = 7;
auto& Func() { return n1;}
auto& r1 = Func();
```

타입 추정 문법에 따라 컴파일러가 참조나 값의 타입을 제대로 선택하게 하려면 `decltype`을 사용하면 된다. 다음 코드는 타입 선택에서 원하는 바는 이루었지만 `Func()`을 반복해서 적어야 하는 불편함이 있다. 이때 `decltype(auto)`를 사용하면 좋다.

```
int n1 = 7;
auto& Func() { return n1;}
decltype(Func()) r1 = Func();
```

`decltype(auto)`를 사용하면 앞의 코드는 다음과 같이 바뀐다.

```
int n1 = 7;
auto& Func() { return n1;}
decltype(auto) r1 = Func();
```

1.2 2진수 리터럴

C++11까지는 정수 리터럴에 8, 10, 16진수가 있었다.

[코드 1-3] C++11의 8, 10, 16진수 정수 리터럴

```
int n1 = 0377; // 8진수. 앞이 0으로 시작.
int n2 = 212;  // 10진수
int n3 = 0x11; // 16진수. 앞이 0x 혹은 0X로 시작.
```

C++14에서는 여기에 2진수 리터럴(binary literal)이 추가되었다. [코드 1-3]의 16진수와 비슷하게 숫자 시작을 `0b` 혹은 `0B`로 시작하면 2진수 리터럴이고 숫자

는 0과 1만 사용할 수 있다.

```
int n1 = 0b1000; // 10진수로 8
int n2 = 0B1000; // 10진수로 8
```

2진수 리터럴 덕분에 비트열 계산을 하는 코드는 이전보다 가독성이 더 좋아진다.

[코드 1-4] 2진수 리터럴을 사용한 비트 플래그

```
#include <iostream>
using namespace std;
int main()
{
    const int ITEM_SKILL_1 = 0b00000001;
    const int ITEM_SKILL_2 = 0b00000010;
    const int ITEM_SKILL_3 = 0b00000100;
    const int ITEM_SKILL_4 = 0b00001000;
    const int ITEM_SKILL_5 = 0b00010000;
    const int ITEM_SKILL_6 = 0b00100000;
    const int ITEM_SKILL_7 = 0b01000000;
    const int ITEM_SKILL_8 = 0b10000000;

    int ItemSkill = 0;

    ItemSkill |= ITEM_SKILL_1;
    cout << "Item Skill: " << ItemSkill << endl;

    ItemSkill |= ITEM_SKILL_3;
    cout << "Item Skill: " << ItemSkill << endl;

    ItemSkill |= ITEM_SKILL_7;
    cout << "Item Skill: " << ItemSkill << endl;

    // ITEM_SKILL_1을 가졌는지
    if((ItemSkill & ITEM_SKILL_1) == ITEM_SKILL_1)
    {
        cout << "ITEM_SKILL_1 소유" << endl;
    }

    // ITEM_SKILL_3을 가졌는지
```

```

if((ItemSkill & ITEM_SKILL_3) == ITEM_SKILL_3)
{
    cout << "ITEM_SKILL_3 소유" << endl;
}

// ITEM_SKILL_5를 가졌는지
if((ItemSkill & ITEM_SKILL_5) != ITEM_SKILL_5)
{
    cout << "ITEM_SKILL_5 소유하지 않았다" << endl;
}

return 0;
}

```

실행 결과

```

Item Skill: 1
Item Skill: 5
Item Skill: 69
ITEM_SKILL_1 소유
ITEM_SKILL_3 소유
ITEM_SKILL_5 소유하지 않았다

```

1.3 숫자 구분

변수에 숫자를 대입할 때 큰 숫자는 그 크기가 쉽게 눈에 들어오지 않는다.

```
int TotalMoney = 20000100;
```

일반적인 문서에서 숫자를 표시할 때 쉼표(.)를 사용해 크기를 쉽게 구분할 수 있는데, 앞의 20000100보다 20,000,100이 훨씬 더 눈에 잘 들어온다.

```
TotalMoney: 20,000,100
```
