C言語
逆引きハンドブック

林 晃◆著

開発現場で役立つプログラミングの常套テクを網羅的に解説!

C言語の機能を目的から探せる!

Windows/
Mac OS X/
Linux(Ubuntu)
に対応!

24時間無料でサンプルデータをダウンロードできます。

C&R研究所

C言語
逆引きハンドブック

林 晃◆著

C&R研究所

■権利について

- Microsoft、Windows、Visual Studio、Visual C++は米国Microsoft Corporationの米国およびその他の国における登録商標または商標です。
- Objective-C、Mac OS、iPhone、Xcodeは、米国および他の国々で登録されたApple Inc.の商標です。
- Linuxは、Linus Torvaldsの米国およびその他の国における登録商標または商標です。
- Ubuntuは、Canonical Ltd.の商標または登録商標です。
- その他、本書に記述されている社名・製品名などは、一般に各社の商標または登録商標です。
- 本書では™、©、®は割愛しています。

■本書の内容について

- 本書は著者・編集者が実際に操作した結果を慎重に検討し、著述・編集しています。ただし、本書の記述内容に関わる運用結果にまつわるあらゆる損害・障害につきましては、責任を負いませんのであらかじめご了承ください。
- 本書で想定している開発環境については、4ページを参照してください。他の環境では、画面のデザインや操作、実行結果などが異なる場合がございますので、あらかじめご了承ください。

■サンプルについて

- 本書で紹介しているサンプルは、C&R研究所のホームページ(http://www.c-r.com)からダウンロードすることができます。ダウンロード方法については、4ページを参照してください。
- サンプルデータの動作などについては、著者・編集者が慎重に確認しております。ただし、サンプルデータの運用結果にまつわるあらゆる損害・障害につきましては、責任を負いませんのであらかじめご了承ください。
- サンプルデータの著作権は、著者及びC&R研究所が所有します。許可なく配布・販売することは堅く禁止します。

●本書の内容についてのお問い合わせについて

この度はC&R研究所の書籍をお買いあげいただきましてありがとうございます。本書の内容に関するお問い合わせは、FAXまたは郵送で「書名」「該当するページ番号」「返信先」を必ず明記の上、次の宛先までお送りください。お電話や電子メール、または本書の内容とは直接的に関係のない事柄に関するご質問にはお答えできませんので、あらかじめご了承ください。

〒950-3122 新潟県新潟市北区西名目所4083-6　株式会社 C&R研究所　編集部
FAX 025-258-2801
「C言語逆引きハンドブック」サポート係

PROLOGUE

スマートフォンやパソコンに限らず、現代のさまざまなハードウェアはソフトウェアの力を組み合わせてより高機能、より魅力的な機器を目指して日々開発が行われています。そして、ソフトウェアに求められている内容も、日々、高度になっていっています。組み込み分野で使われるファームウェア、パソコン上で動作するデバイスドライバやアプリケーション、これらはまったく異なるもののように見えますが、その開発にはいずれもC言語が使われることが多くあります。現在のソフトウェア開発では、C言語だけですべてが解決するのではなく、各開発対象に合わせたプログラミング言語を組み合わせますが、その中でもC言語は広く汎用性があり、ハードウェアに近づくほど活躍する場面が多くなる言語です。

本書は、アプリケーションではなく、C言語の言語としての機能や標準ライブラリ、一部のシステムコールについて解説しています。本書を執筆している時点で正式に規格化されている最新規格はC99ですが、その規格の内外にこだわらず、C言語を使った開発で使用されることが多いものについて解説しています。

本書は「C言語ではどのように書くのか?」ということから使用する関数や書き方を調べることができる逆引きリファレンスです。解説している関数を呼び出すとどのように動作するのか、何が返ってくるのか、ということがわかるように、各項目のサンプルコードは完結したプログラムになっています。各サンプルコードはできるだけシンプルになるようにしています。また、Visual C++、Xcode、Ubuntuの3つの環境で動作するサンプルコードにし、どうしても同じコードにできないところについては、Visual C++ではどのように書き、XcodeやUbuntuではどのように書くのかということもわかるように、コードを併記しています。

CHAPTER 01ではC言語の特徴や本書で使用する開発環境について解説しました。CHAPTER 02ではC言語の文法について解説しました。CHAPTER 03以降では分野ごとにC言語の標準ライブラリやシステムコールについて解説しています。気になるところから読んでいただいてもよいと思います。

C言語を使って開発する分野は非常に広く、本書で解説している関数以外にも、開発対象のシステムコールや、ハードウェア、画像処理や音声処理などのアルゴリズムなど、多くの知識が必要になると思います。しかし、開発するソフトウェア独自の部分では必ず、本書で解説している関数や機能を使用することがあると思います。本書がC言語の学習や開発の現場で役に立つことができれば幸いです。

最後に、本書を執筆するにあたってお世話になったスタッフの皆様に心から感謝申し上げます。そして読者の皆様のお役に立つことができれば、著者として、これ以上の幸せはありません。読者の皆様が開発したソフトウェアが、さまざまな分野で活躍していくことを願っております。

2011年9月

アールケー開発 代表　林 晃

本書について

開発環境について

本書で想定している開発環境は、次の通りです。

- OS：Windows 7 / 処理系：Visual C++ 2010 Express
- OS：Mac OS X 10.7.x / 処理系：Xcode 4.1.1
- OS：Ubuntu 11.04 / 処理系：GCC 4.5.2

本書の表記について

本書の表記についての注意点は、次のようになります。

▶エスケープ記号について

本書ではエスケープ記号の表記に「\」（バックスラッシュ）を使用していますが、日本語版のWindowsやUbuntu上ではフォントの関係から「¥」記号になります。

▶関数のプロトタイプ宣言について

関数のプロトタイプ宣言は、開発環境の違いやバージョン、設定により、型や修飾子の有無については厳密には異なる場合があります。また、C99から標準ライブラリの関数の中には「restrict」修飾子が追加されているものがありますが、対象としている開発環境で共通していないため、本書では「restrict」修飾子は省略しています。

▶サンプルコードの中の▼について

本書に記載したサンプルコードは、誌面の都合上、1つのサンプルコードがページをまたがって記載されていることがあります。その場合は▼の記号で、1つのコードであることを表しています。

サンプルファイルのダウンロードについて

本書のサンプルデータは、C&R研究所のホームページからダウンロードすることができます。下記のURLにアクセスし、表示されたページの「サンプルのダウンロード」のアイコンをクリックしてください。

URL **http://www.c-r.com/mo_clang.htm**

なお、サンプルデータをダウンロードするときには、下記に記載のユーザー名とパスワードが必要になります。

サンプルのダウンロードに必要なユーザー名とパスワード

ユーザー名 **cggh**

パスワード **cgr92**

※ユーザー名・パスワードは、半角英数字で入力してください。

サンプルコードの利用方法

サンプルファイルは、開発環境別に3つのフォルダに分かれており、開発環境別のフォルダ内は、CHAPTERごとのフォルダの中に項目番号のフォルダに分かれています。COLUMNのサンプルについては、「項目番号-COLUMN」（COLUMNのサンプルが複数ある場合は、末尾に連番）としています。サンプルはZip形式で圧縮してありますので、解凍してお使いください。

それぞれのフォルダ内には、プロジェクトファイルとソースファイルが保存されています。Visual C++では、拡張子「.sln」のソリューションファイルをダブルクリックして、Visual C++で開き、[デバッグ(D)]メニューから[デバッグ開始(S)]を選択するとプログラムが作成され、実行されます。実行結果は、自動的に起動するコンソールに出力されます。

Xcodeでは、拡張子「.xcodeproj」のプロジェクトファイルをダブルクリックしてXcodeで開き、[Product]メニューから[Run]を選択するとプログラムが作成され、実行されます。実行結果は、「Console」エリアに出力されます。「Console」エリアは自動的に表示されます。表示されない場合には、[View]メニューから[Debug Area]→[Activate Console]を選択すると表示されます。

Ubuntuでは、ソースファイルのみが格納されています。端末から直接、「GCC」プログラムを使ってソースファイルをビルドします。端末でソースファイルをビルドするには、各ソースファイルが格納されているディレクトリを「cd」コマンドでカレントディレクトリに設定（「cd ディレクトリパス」と入力）し、「gcc ソースファイル名 -o 出力プログラム名」と入力します。ビルドが成功するとプログラムがカレントディレクトリに出力されるので、端末から「./出力プログラム名」と入力して、プログラムを起動します。プログラムの出力は、起動した端末に出力されます。なお、ソースファイルによっては、ビルドするときに追加オプションが必要な場合があります。必要なオプションについてはその都度、本書の「HINT」に記載しています。

▶Xcodeの設定について

Xcodeのサンプルファイルは、Mac OS X 10.7上で作成しています。そのため、プロジェクトファイルのデフォルト設定が、Mac OS X 10.7以降で動作するように設定されているため、Mac OS X 10.6上で動作しているXcodeでは警告が表示されます。次のように操作してプロジェクトファイルの設定を変更してください。

❶ ソリューションウインドウのでナビゲータエリアからプロジェクトファイルをクリックします。

❷ プロジェクトの設定の編集画面が表示されるので、編集画面のリストから「PROJECT」の中に表示された、プロジェクトファイルを選択します。

❸ 「Build Settings」タブの「Mac OS X Deployment Target」の値を「Mac OS X 10.6」に変更します。

上記の手順によりMac OS X 10.6上でもビルドできるようになります。

CONTENTS

CHAPTER 01 C言語の基礎知識

CHAPTER 02 C言語の文法

CHAPTER 04 メモリブロックと文字列

CHAPTER 05 数学処理

CHAPTER 01

C言語の基礎知識

C言語とは

C言語とは

C言語は、オペレーティングシステムの開発にも使用することができる手続き型言語です。C言語は1972年にAT&Tのベル研究所で開発されました。当時はUNIX向けに開発されましたが、現在ではさまざまな環境に移植され、PC上で動作するソフトウェアの開発だけではなく、組み込み系やiPhoneアプリケーションの開発にも使用することができる汎用性の高いプログラミング言語です。

▶軽快な動作

一般的なC言語の処理系は、ソースファイルをコンパイラでコンパイルし、ネイティブコードに変換してプログラムを作成します。実行時にソースファイルの内容を解釈する、インタプリタ型の言語と比較して動作が軽快です。

ネイティブコードに直接、変換しない処理系としては、Microsoft社が開発した「.NET」があります。「.NET」ではC言語のソースファイルをコンパイルし、中間言語に変換します。中間言語に変換されたプログラムは、実行時に、動作環境に最適化したネイティブコードに変換されて実行されます。

C言語の処理系は、コンパイラ型が一般的ではありますが、インタプリタ型でC言語のコードを実行する処理系もあります。これには、「CINT」などがあります。

- Microsoft .Netホーム
  URL http://www.microsoft.com/japan/net/
- CINT
  URL http://root.cern.ch/drupal/content/cint

▶低レベルな処理の記述

C言語はオペレーティングシステム(OS)も記述することができる言語です。ハードウェアに近い、非常に低レベルな処理も記述することができ、ハードウェアの制御やメモリの操作なども行うことができます。そのため、デバイスドライバの開発やハードウェアに組み込まれるファームウェアの開発にも使用することができ、幅広い分野で利用されています。

▶プログラミング言語そのものの開発にも使用できる

低レベルな処理を記述することができるC言語は、プログラミング言語そのものの開発にも使用することができます。C言語のコンパイラとしても使われるGCC(the GNU Compiler Collection)の開発にもC言語が使われており、最近では一部、C++も使われているようです。

▶C++やObjective-Cとの連携

C言語はC++やObjective-Cなど、C言語がベースになっているプログラミング言語とも組み合わせることができます。C言語で記述された関数は、これらのプログラミング言語から直接、呼び出すことができます。呼び出すだけではなく、これらのプログラミング言語で記述されたソースファイルの中で、直接、C言語の関数を記述することもできます。

▶豊富な動作環境

C言語のコンパイラは非常に多くのプラットフォームに移植されているため、幅広いプラットフォームで使用することができます。それぞれのプラットフォームには、専用の機能を呼び出すためのシステムコールが用意されていますが、それらを使用していないコードは、そのまま移植可能です。これは、C言語は国際標準化機構（ISO）により規格化されており、各コンパイラはその規格に準拠しているためです。

▶標準ライブラリ

C言語には、さまざまな機能を提供する標準ライブラリがあります。標準ライブラリにはファイルの入出力や数学処理、日時の処理などを行うための関数が含まれています。組み込み系の開発やデバイスドライバの開発などでは使用できないケースもありますが、C言語が使用可能な多くのプラットフォームで利用することができます。これにより、システムコールを使用しない処理については、プラットフォームを意識せずに移植性の高いコードを記述することができます。

▶共通システムコール

通常、システムコールは各プラットフォームにより異なるものですが、POSIX規格と呼ばれる共通のシステムコールがあります。これは、ISOにより定義されるC言語の規格や標準ライブラリには含まれない部分を定義した規格です。これにより、POSIX規格をサポートしたプラットフォーム間では、プログラムコードに互換性を持たせることができます。POSIX規格は多くのUNIX系のシステムでサポートされており、iOSやMac OS Xでも使用することができます。

▶コンパイル時のテキスト処理

C言語にはプリプロセッサディレクティブという機能があります。この機能を使用すると、コンパイル時にソースファイルを書き換えることができます。書き換えるとはいっても、実際にファイルの内容を変更して保存してしまうというものではなく、コンパイル時にコンパイラに読み込まれるコードを書き換える機能なので、安心して使用できます。たとえば、Windows上でコンパイルするときだけ、特定の関数を定義するということができます。他にも複数のソースファイルから共通して使用される定義をヘッダファイルに記述し、それを各ソースファイルから読み込んで使用することもできます。

▶静的なエラー検出

C言語は動的な型付けを行うプログラミング言語でないので、他の動的言語のような柔軟性はありません。しかし、だからこそコンパイル後に行われる、リンク時に必要な関数がすべて定義されているかを検出することができます。システムコールの中には実行時でなければ使用可能かどうかわからないものもありますが、スペルミスのような関数名の記述ミスはリンクエラーとなるので、記述ミスしたことが実行時まで待たなくともわかります。コンパイラの設定によっては、コンパイル時にエラーとして検出され、どのソースファイルのどの行が間違っているのかもわかります。

C言語の規格について

C言語に関する規格の種類について

C言語は国際標準化機構（International Organization for Standardization。以降、ISOと記述）により規格化されていますが、ISOの規格以外にもいくつか関連する規格があります。関連する代表的な規格にはISOも含めて次のようなものがあります。

- ISO/IEC 9899:1990
- ISO/IEC 9899:1999
- ISO/IEC 9899:201X
- IEEE Std 1003.1（POSIX.1）
- Single UNIX Specification（SUS）

上記以外にも、ISOの規格の元になっている米国規格協会（American National Standards Institute。以降、ANSIと記述）による規格や日本工業規格（Japanese Industrial Standards Committee。以降、JISと記述）での規格などもあります。

▶「ISO/IEC 9899」について

「ISO/IEC 9899」は、C言語の言語としての規格と標準ライブラリについて定義している規格です。上記の通り、本書の執筆時点では3つの規格があります。「ISO/IEC 9899:1990」は通称「C90」と呼ばれる規格です。「ISO/IEC 9899:1999」は通称「C99」と呼ばれる規格です。「ISO/IEC 9899:201X」は通称「C1X」と呼ばれる規格です。これらの中で、本書の執筆時点で正式に規格化されている最新の規格は「ISO/IEC 9899:1999」です。「ISO/IEC 9899:201X」はドラフト規格であり、まだ、規格化に必要な作業が続いています。

「ISO/IEC 9899:1999」については、次のように更新が3回行われています。

- 2001年 TC1（Technical Corrigendum 1）
- 2004年 TC2（Technical Corrigendum 2）
- 2007年 TC3（Technical Corrigendum 3）

最新の規格は、「ISO/IEC 9899:1999」に、これら3回の更新を反映したもので、「WG14 N1256」という名称でドキュメント化されています。詳しくは、次のWebページを参照してください。

- ISO/IEC JTC1/SC22/WG14 - C: Approved standards

  URL http://www.open-std.org/jtc1/sc22/wg14/www/standards

▶「IEEE Std 1003.1」について

「IEEE Std 1003.1」は「POSIX.1」と呼ばれる規格です。異なるシステム間での共通のシステムコールやヘッダファイルを定義しています。これにより「POSIX.1」に準拠したシステム間ではソースコードに互換性があり、移植が容易になるというメリットがあります。ただし、「POSIX.1」に準拠していても、定義されているシステムコールの対応度合いはシステムによ

り異なるので、移植対象のシステムのドキュメントを確認する必要があります。

なお、「POSIX.1」は共通のシステムコールやヘッダファイルを定義した規格で、「POSIX」規格の一部です。「POSIX」規格では、より広範囲な事柄を定義しています。

Windows上で「POSIX.1」で定義されているシステムコールを使用するには、POSIXサブシステムを別途、インストールする必要があります。

Unix系のシステムやLinux、Mac OS X、iOS(iPhone OS)などは、標準で「POSIX.1」がサポートされています。

▶ Single UNIX Specification(SUS)

UNIXシステムの標準規格全体を定義している一連の規格です。C言語の関数やヘッダファイルだけではなく、UNIXシステムに関するさまざまな事柄を定義しています。この規格に準拠したシステムだけが「UNIX」を名乗ることができます。本書の執筆時点での最新版は、「Version 4」です。たとえば、Mac OS Xでは、Mac OS X 10.5 Leopardから「SUSv3(Single UNIX Specification Version 3)」に準拠し、はじめて「UNIX」を名乗ることができるシステムになりました。「SUS」に準拠していないシステムは、「UNIX」に近くとも「UNIX」とは名乗ることができず、「Unix系」と呼ばれます。

▶ 3種類の規格の関係について

「ISO/IEC 9899」「POSIX.1」「SUS」の3種類の規格により定義されている関数は、その定義する目的や範囲の違いにより、この記述順に、前者が後者を含むような関係があります。

開発環境について

C言語での開発の流れについて

C言語を使ったソフトウェアの開発は、おおよそ次のような流れで行います。

1 プロジェクトを作成する。
2 ソースコードを記述する。
3 プログラムをビルドする。
4 動作テストを行う。
5 不具合が見つかったら、デバッガなどを使用して不具合の原因を調べる。
6 不具合を修正する。

2から6までのステップは何度も繰り返し、ソフトウェアを開発します。また、使用する開発環境によっては1は行わず、「Makefile」ファイルなど、プログラムをビルドするために必要なファイルを作成する場合もあります。

開発環境の種類

プログラムを作成するために必要なプログラム一式を「開発環境」と呼びます。開発環境には、ソースコードをコンパイルするための「コンパイラ」、コンパイラによって作成されたオブジェクトコードとライブラリをリンクさせてプログラムをビルドするための「リンカ」、ソースコードを入力するための「テキストエディタ」、プログラムをデバッグするための「デバッガ」などが含まれます。その他、さまざまなプログラムで構成されています。

Mac OS XやWindowsでは、統合開発環境(IDE)と呼ばれる開発環境を使用することが一般的です。統合開発環境はコンパイラやテキストエディタ、リンカ、デバッガなどを統合的に扱う機能を持っており、効率的に開発を行うことができます。

本書では、次の統合開発環境を使用します。

- Visual C++ 2010 Express(Windows)
- Xcode(Mac OS X)

なお、本書では、Ubuntu上では統合開発環境は使用せず、GCCやGDBというコマンドラインツールを直接、使用します。

Visual C++ 2010 Expressについて

Visual C++ 2010 Expressは、Visual Studio 2010 ExpressのC言語およびC++用の開発環境です。Visual StudioはWindows上で動作する統合開発環境で、複数のエディションがあります。Expressは、評価や学習、ホビー向けに無償で提供されています。他のエディションは有償となりますが、Expressにはない、さまざまな機能やサービスが提供されます。詳

しくは、次のWebサイトを参照してください。

- Microsoft Visual Studio ホームページ

URL http://www.microsoft.com/japan/msdn/vstudio/

▶ Visual C++ 2010 Expressのセットアップ

Visual C++ 2010 Expressを、Microsoft社のWebサイトからダウンロードします。

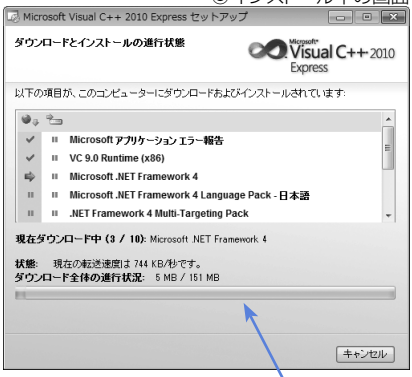
- Microsoft Visual Studio Express

URL http://www.microsoft.com/japan/msdn/vstudio/express/

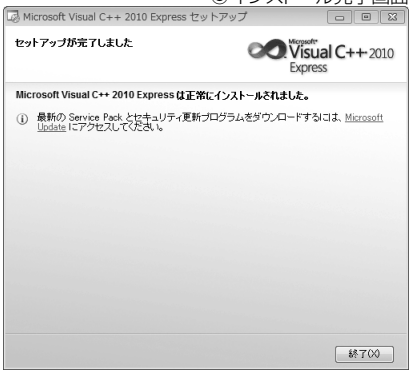
インストール方法には、Webインストールとオフラインインストールがあります。Webインストールは必要なファイルをダウンロードしながらインストールする方法で、オフラインインストールはインストーラ全体をダウンロードしてからインストールする方法です。本書では、Webインストールを使用します。

まず、Visual Studio 2010 ExpressのページからVisual C++のWebインストール用のインストーラをダウンロードします。次にダウンロードしたインストーラを起動し、画面の指示に従ってインストールします。

◉インストール中の画面



◉インストール完了画面



画面の指示に従ってインストールする

インストールが完了すると、「スタート」メニューの「すべてのプログラム」に「Microsoft Visual Studio 2010 Express」フォルダが作成されます。このフォルダの中の「Microsoft Visual C++ 2010 Express」を選択すると、インストールしたVisual C++を起動することができます。

ダウンロードしたインストーラは、インストール後は必要ないので削除します。

▶ 製品の登録について

Visual C++ 2010 Expressは無償ですが、使い続けるためには登録が必要です。登録は無料です。未登録の状態では日数制限があり、スプラッシュ画面には「評価目的でのみ使用可能です」と表示されます。登録するには、[ヘルプ(H)]メニューから[製品の登録(P)]を選択します。登録ダイアログボックスが表示されるので、[オンラインで登録キーを取得する(O)]ボ

タンをクリックし、画面の指示に従って登録します。登録が完了すると登録キーを取得することができます。取得した登録キーを登録ダイアログボックスに入力して登録作業は完了です。

▶拡張子の表示について

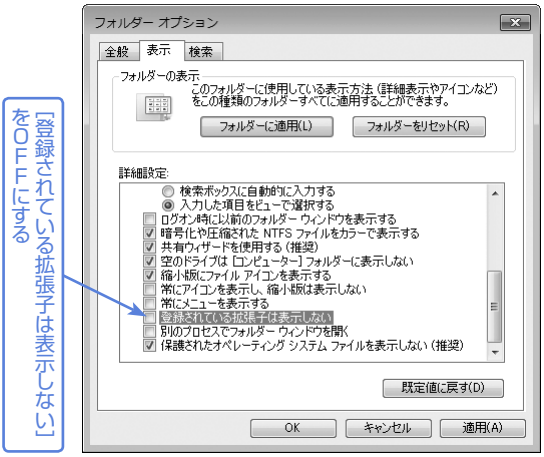
Windowsは初期状態では拡張子を表示しないようになっています。ソフトウェア開発では多くのファイルを使用しますが、ソースファイルとヘッダファイルや、プロジェクトファイルとソリューションファイルのように、ファイル名の拡張子だけが異なるようにする場合が多く、拡張子が表示されていないと何かと不便です。Windows 7では、次のように操作して拡張子が表示されるように設定を変更します。Windows VistaやWindows XPでも、多少、メニューの文字列が異なりますが、ほぼ同様の方法で設定可能です。

❶「スタート」メニューから[コンピュータ]を選択します。

❷ [整理]メニューから[フォルダーと検索のオプション]を選択します。「フォルダーオプション」ダイアログボックスが表示されます。

❸「表示」タブをクリックします。

❹ [詳細設定]から[登録されている拡張子は表示しない]をOFFにします。



❺ [OK]ボタンをクリックします。

Xcodeについて

Xcodeはアップル社が提供する統合開発環境です。Mac OS X上で利用可能です。XcodeはコンパイラにはGCCやLLVM、デバッガにはGDBを使用しています。Xcodeはこれらのツールのフロントエンドとして動作し、Xcode上からプログラムをビルドしたり、デバッグしたりすることができます。本書の執筆時点での最新版はXcode 4.1です。Xcode 4.1は、Mac OS X 10.7用とMac OS X 10.6用があります。Mac OS X 10.7ではMac App StoreからXcode 4.1を無償でダウンロードできます。Mac OS X 10.6では、Xcode 4.1またはXcode 3を使用します。どちらもMac Dev ProgramやiOS Dev Programの有料会員は専用のWebサイトからダウンロードできます。

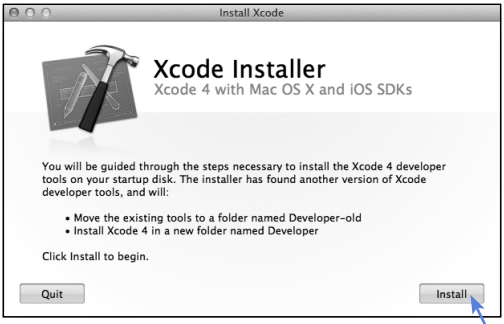
Mac App StoreからXcodeをダウンロードするには、Apple IDが必要になります。Apple IDの登録は無料です。ADCのアカウントや、iTunes Store、Apple Storeなどで使用しているApple IDをすでに持っている場合は、そのApple IDを使用することもできます。

本書では、執筆時点での最新のXcode 4.1を使って解説します。

▶ Xcodeのセットアップ

Xcodeをインストールします。「App Store」を起動します。一度もサインインをしたことがないマシンやサインアウトしている場合には、先に右側に表示されている「ナビリンク」の「サインイン」をクリックしてApple IDでサインインしてください。検索ボックスに「Xcode」と入力します。Xcodeが検索結果に表示されるので、「インストール」をクリックします。Xcodeのダウンロードが始まります。

ダウンロードが完了したら、ダウンロードされた「Install Xcode」を起動します。



「Install」ボタンをクリックして、画面の指示に従ってインストールする

インストーラが起動したら[Install]ボタンをクリックし、画面の指示に従ってインストールを行います。インストールが完了すると、ルートディレクトリの「Developer」フォルダ内に開発ツールやSDK、ドキュメントなどがインストールされ、Xcodeが起動します。次回からXcodeを起動するときは、「/Developer/Applications」フォルダ内にインストールされた「Xcode」を起動します。

Ubuntuについて

UbuntuはLinuxのディストリビューションの1つで、無償で提供されるOSです。デスクトップ版とサーバ版があり、定期的にセキュリティアップデートなどが行われており、日本語環境を簡単に導入できる「Ubuntu Desktop 日本語 Remix CD」が提供されるなど、導入しやすいディストリビューションです。

Ubuntuについての詳細は、次のWebサイトを参照してください。

- Ubuntu Japanese Local Community Team

  URL http://www.ubuntulinux.jp/

▶Ubuntuのセットアップ

「Ubuntuの入手」ページから日本語 Remix CDのCDイメージファイルをダウンロードし、CD-Rに書き込んでインストールします。本書では、執筆時点で最新版の「ubuntu-ja-11.04-desktop-i386.iso」ファイルを使用します。異なるバージョンのイメージファイルを使用する場合には、ファイル名や文字列が一部、異なる可能性があります。ご使用になる環境に合わせて読み替えてください。

- Ubuntuの入手

  URL http://www.ubuntulinux.jp/products/GetUbuntu

なお、Ubuntuは完全なOSであり、Windows上やMac OS X上で動作するソフトウェアではありません。そのため、他のOSとのデュアルブートはできますが、基本的にはインストール先のハードディスクを消去したり、パーティションで分離してインストールします。すでにWindowsがインストールされているマシンにインストールしたい場合には、「Wubi」を使用するとWindows上で簡単にUbuntuを導入することができます。「Wubi」については、次のWebサイトを参照してください。

- Wubi - Ubuntu Installer for Windows

  URL http://wubi-installer.org/

また、Ubuntuを使ってみたいが、既存のシステムには影響を与えたくないという場合には仮想PCを利用するのがよいでしょう。

Ubuntuをインストールするには、ダウンロードしたイメージファイルをCD-Rに書き込み、そのCD-Rからインストール先のマシンを起動します。起動すると自動的にインストール画面になります。画面の指示に従ってインストールしてください。

途中で「ディスク領域の割り当て」という画面が表示されます。この画面は内蔵ハードディスクをどのようにパーティションに区切るかを設定する画面です。すでにWindowsがインストールされているマシンにセットアップする場合と、何もインストールされていないマシンにセットアップする場合とで、異なる画面が出ます。画面の指示に従って、セットアップしたい構成を選択します。

インストールが完了したら、インストールの途中で設定したアカウントを使用してログインします。アカウント設定時に「自動的にログインする」を選択していると、ログイン画面は表示されずに、自動的にログインされます。

▶開発環境について

Ubuntuでは、開発に必要なC言語のコンパイラやデバッガ、テキストエディタが標準でインストールされています。本書では、コンパイラはGCCを使用し、デバッガはGDB、テキストエディタはgeditを使用します。

Ubuntu 11.04からは「Unity」という新しいグラフィックインターフェイスが採用されています

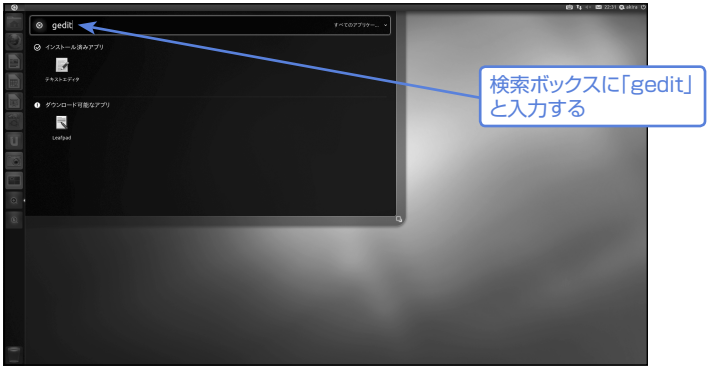
が、どちらのグラフィックインターフェイスでもコンパイルやデバッグの手順は変わりません。異なるのは、端末やテキストエディタの起動方法です。

geditは、次のようにして起動します。

● Unityの場合

❶ サイドバーから「アプリケーション」ボタンをクリックします。

❷ 検索ボックスに「gedit」と入力します。



❸ 「インストール済みアプリ」に「テキストエディタ」が表示されるので、表示された「テキストエディタ」をクリックします。

● Ubuntu Classicの場合

❶ [アプリケーション]メニューから[アクセサリ]→[テキストエディタ]を選択します。

コンパイラやデバッガを呼び出すのに使用する端末は、次のようにして起動します。

● Unityの場合

❶ サイドバーから「アプリケーション」ボタンをクリックします。

❷ 検索ボックスに「terminal」と入力します。

❸ 「インストール済みアプリ」に「端末」が表示されるので、表示された「端末」をクリックします。

● Ubuntu Classicの場合

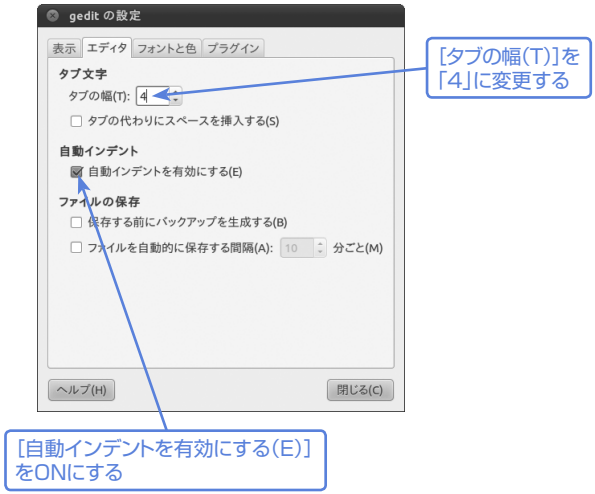
❶ [アプリケーション]メニューから[アクセサリ]→[端末]を選択します。

また、geditは、初期状態の設定では、プログラムコードを入力するのには向かない設定になっています。次のように設定します。

❶ [編集(E)]メニューから[設定(E)]を選択します。

❷ 「エディタ」タブをクリックします。

❸ [タブの幅(T)]を「4」に変更し、[自動インデントを有効にする(E)]をONにして、[閉じる(C)]ボタンをクリックします



コードの記述方法

ソースファイルの作成方法

プログラムコードを記述するソースファイルは、他のプログラミング言語と同様に、テキストエディタや統合開発環境で記述します。使い慣れているテキストエディタがあれば、それを使うこともできます。お勧めは統合開発環境です。統合開発環境はプログラムコードを記述するだけではなく、ビルドやデバッグも行うことができるので、効率よく開発を進めることができます。本書では、Windows上ではVisual C++ 2010 Express、Mac OS X上ではXcodeを使用します。

Ubuntuについては、統合開発環境は使用せず、C言語の構文に対応したgeditというテキストエディタを使用します。

拡張子について

C言語ではソースファイルとヘッダファイルの2種類のファイルを使用します。ソースファイルの拡張子は「.c」です。ヘッダファイルの拡張子は「.h」を使用します。また、ファイル名には日本語などのマルチバイト文字は使用しないようにし、スペースなども含まないようにします。

テキストエンコーディングについて

使用できるテキストエンコーディングは、コンパイラに依存します。本書で使用する、GCCやXcode（Xcodeのコンパイラは設定により、GCCやLLVMが使われる）、Visual C++では、任意のテキストエンコーディングを使用可能です。日本語環境のWindows上で開発する場合には、シフトJISが使われることが多いようです。XcodeではUTF-8（BOMなし）を使用することが多く、Linux上でも最近はUTF-8を使用することが多いようです。以前は、Linux上ではEUC-JPが多く使われていました。著者の経験では、Windows、Mac OS X、Linuxなどのそれぞれをまたがって共通で使用するソースファイルでは、UTF-8（BOMなし）を使用するとトラブルが少なく、お勧めです。

▶日本語について

UTF-8やシフトJIS、EUC-JPなど、日本語に対応したテキストエンコーディングを使用すればコード内のコメントに日本語を使用できます。しかし、プログラムで使用する文字列として、直接、日本語を記述すると、コンパイラの組み合わせによっては文字化けしてしまいます。Mac OS X上で使用するGCCやLLVMでは、UTF-8を使用しているときには直接、日本語の文字列を記述することも可能ですが、他の開発環境では、開発環境のバージョンなどにも依存してしまいます。通常は、日本語を含む文字列は、ソースファイル中では直接、記述せず、リソースファイルなどの外部ファイルやリソースデータなどに記述しておき、それを読み込むようにする方が安全です。リソースファイルやリソースデータの扱い方は、各OSなどに依存してくるので、それぞれのプラットフォームのドキュメントを参照してください。

はじめてのC言語のプログラム

「HelloWorld」プログラムの作成

さまざまなプログラミング言語がありますが、どのプログラミング言語でも最初の例題として取り上げるプログラムに「HelloWorld」というプログラムがあります。「HelloWorld」は名前の通り、「Hello World!」という文字列を表示するだけのプログラムです。本書でもこのプログラムを最初に解説したいと思います。また、このプログラムの作成を通して、各開発環境の操作方法を解説します。コードは共通で、次のコードを入力します。

SAMPLE CODE 「HelloWorld.c」のコード

```
/* HelloWorld.c
   HelloWorldプログラムのソースコード
*/

#include <stdio.h>

#ifdef _MSC_VER
#include <conio.h>
#endif

/* メイン関数 */
int main(int argc, char *argv[])
{
    /* 文字列を出力する */
    printf("Hello World!\n");

#ifdef _MSC_VER
    /* キーが押されるまで待機する */
    _getch();
#endif

    /* エラーなし */
    return 0;
}
```

▶コメントについて

「/*」から「*/」で囲まれた範囲はコメントです。この部分はコンパイラによって無視されます。詳しくは、《**コメントについて**》(p.62)を参照してください。

▶「#include」について

「#include」文はヘッダファイルを読み込むための命令文です。ここでは「stdio.h」ファイルと「conio.h」ファイルを読み込んでいます。「conio.h」ファイルはVisual C++でのみ使用可能なヘッダファイルです。そのため、Visual C++でのみ読み込まれるようにプリプロセッサ

ディレクティブの「#ifdef」を使用しています。詳しくは、《**プリプロセッサディレクティブについて**》(p.129)を参照してください。

▶「main」関数について

「main」関数は、プログラムのスタート地点になる関数です。プログラムが起動すると、「main」関数が実行されます。「main」関数は、2つの引数と「int」型の戻り値を持ちます。最初の引数「argc」はコマンドラインパラメータの個数が入り、2番目の引数「argv」はコマンドラインパラメータが文字列の配列として格納されています。配列についての詳細は、《**配列について**》(p.99)を参照してください。

たとえば、「Sample」というプログラムが、コマンドラインから次のように入力されて起動したとします。

```
./Sample -f abc.txt
```

このとき、引数「argc」の値は「3」となり、引数「argv」には次のような配列が格納されています。

配列のインデックス番号	配列の要素の内容
0	./Sample
1	-f
2	abc.txt

配列の先頭には必ず、プログラム名が格納されます。そのため、「argv」の値も実際に指定されたオプションの個数よりも1大きい値になります。

戻り値はプログラムの終了コードです。特にエラーが発生しなかったときは「0」を返します。

なお、「main」関数は、規格では次のいずれかの形式で実装するように定義されているので、必ずしも上記の形式になるとは限りません。

```
int main(void);
int main(int argc, char *argv[]);
```

プラットフォームによっては、さらに引数が追加された「main」関数を使用することもあります。

▶「printf」関数について

「printf」関数はコンソールに文字列を出力する関数です。ここでは「Hello World!\n」という文字列を指定しています。C言語の文字列は「"」記号で囲みます。文字列の最後の部分の「\n」は改行を表すエスケープシーケンスです。なお、日本語版のWindowsとUbuntu上では「\n」は「¥n」になります。「printf」関数についての詳細は、《**フォーマットを指定して文字列を出力する**》(p.160)を参照してください。

▶「_getch」関数について

「_getch」関数は何かキーが押されるまで待機し、キーが押されると押されたキーを返す関数です。ここでは何が押されたかは知る必要がないため、単純に呼び出し、関数の戻り値は取得していません。また、この関数はVisual C++専用の関数のため、Visual C++上でのみ、

使用するようにしています。Visual C++では、プログラムが終了するとコンソールウィンドウがすぐに閉じてしまうので、このようにして待機しています。本書で使用する他の開発環境では、コンソールウィンドウに該当するウィンドウがすぐに閉じることはないため、待機する必要はありません。

Visual C++ 2010 Expressの操作方法

Visual C++ 2010 Expressでは、プログラムごとにプロジェクトを作成します。プロジェクトはソリューションに登録して使用します。ソリューションには複数のプロジェクトを登録することができ、関連するプログラムのソースファイルを統合的に管理することができます。プログラムのソースファイルやヘッダファイルはプロジェクトに登録します。登録されたソースファイルは[ソリューションのビルド(B)]コマンドでコンパイルされ、プログラムがビルドされます。

▶ソリューションとプロジェクトの作成

次のように操作し、「HelloWorld」用のソリューションとプロジェクトを作成します。

❶ [ファイル(F)]メニューから[新規作成(N)]→[プロジェクト(P)]を選択します。「新しいプロジェクト」ダイアログボックスが表示されます。

❷ ダイアログボックスの左側のツリーの上部にある「インストールされたテンプレート」が選択されていない場合は選択します。

❸ ツリーから「Visual C++」を選択します。

❹ テンプレートから「Win32 コンソールアプリケーション」を選択します。

❺ [名前(N)]に「HelloWorld」と入力します。

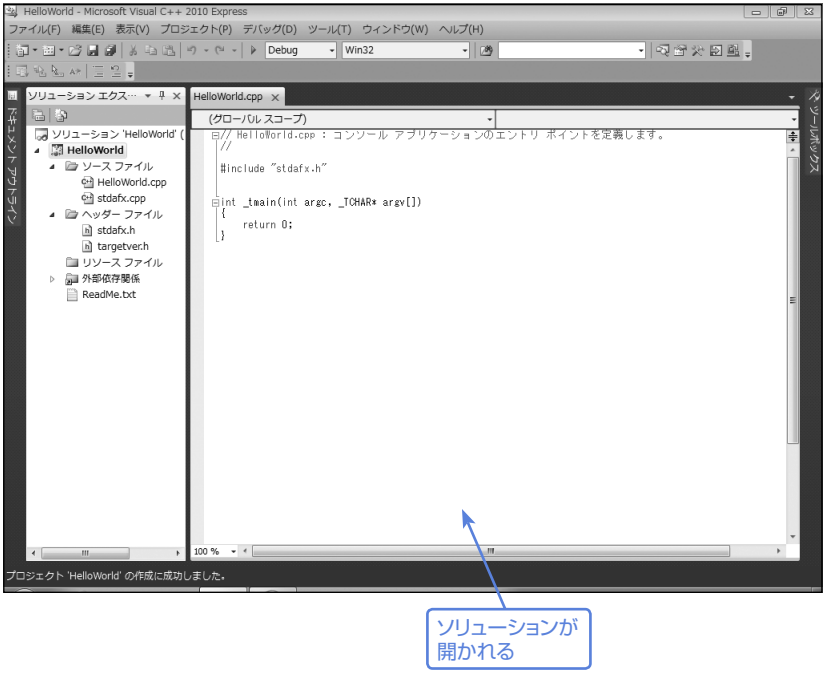
❻ [参照(B)]ボタンをクリックし、ソリューションとプロジェクトの保存先を選択します。

❼ [ソリューションのディレクトリを作成(D)]がONになっていることを確認し、[OK]ボタンをクリックします。このチェックボックスがONになっていると、[場所(L)]に指定したフォルダに、新しいフォルダを作成し、そのフォルダ内にプロジェクトとソリューションを作成します。

❽「Win32 アプリケーション ウィザード」が表示されます。[次へ]ボタンをクリックします。

❾「アプリケーションの種類」から[コンソールアプリケーション(O)]をONにし、[追加のオプション]をすべてOFFにし、[完了]ボタンをクリックします。

❿ 指定したフォルダに「HelloWorld」フォルダが作成され、ソリューションファイルやプロジェクトファイル、ひな形のソースファイルなどが作成されて、ソリューションが開かれます。



ソリューションが開かれると、初期状態では、ウィンドウの左側に「ソリューション エクスプローラー」が表示され、開いているソリューションやそのソリューションに登録されているプロジェクト

が表示されます。「ソリューション エクスプローラー」はツリー表示になっており、プロジェクトの中にはさらにそのプロジェクトに登録されているソースファイルが表示されます。

ウィンドウ右側はテキストエディタになっています。ソリューションエクスプローラーでダブルクリックしたファイルが表示され、この領域で直接、編集し、保存することができます。この内蔵テキストエディタはタブ表示になっています。開いたファイルは次々にタブとして追加されます。テキストエディタ領域の上部にタブが表示されるので、クリックしてファイルを切り替えたり、タブの中の閉じるボタン(「×」ボタン)をクリックして、ファイルを閉じることができます。

▶ コードの入力準備

コードを入力する前に次のように操作してファイル名を変更します。

❶ ソリューションエクスプローラーから「HelloWorld.cpp」を選択します。

❷ 選択した「HelloWorld.cpp」を右クリックし、ショートカットメニューから[名前の変更(M)]を選択します。

❸ ファイル名を「HelloWorld.c」に変更し、「Enter」キーを押します。

名前を変更した理由は、入力するコードがC言語のコードであり、C言語のルールでコンパイルしたいからです。拡張子が「.cpp」のファイルはC++のソースファイルで、C++のルールでコンパイルされます。「stdafx.cpp」はプリコンパイルヘッダファイルを生成するためのファイルです。

プリコンパイルヘッダファイルは、プロジェクトに登録したソースファイルをコンパイルするときに、必ず最初に読み込むヘッダファイルです。コンパイルが終わった状態の中間ファイルとして保存され、各ソースファイルでの読み込み時間が短縮され、コンパイル時間が短くなるという利点があります。通常はシステムコールが定義されているヘッダファイルなど、必ず読み込んでほしいヘッダファイルを読み込むように記述します。プリコンパイルヘッダファイルの内容は、「stdafx.h」に記述します。また、プリコンパイルヘッダファイルの読み込みは、必ずソースファイルの先頭で行う必要があります。そのため、ひな形から作成された「HelloWorld.cpp」は、「stdafx.h」の読み込みが先頭に書かれています。ただし、本書ではプリコンパイルヘッダファイルは使用しないので、コードでも記述しません。

▶ コードの入力

「HelloWorld.c」にコードを記述します。すでにファイルが開かれているはずなので、そのまま入力してください。閉じてしまった場合など、ファイルが表示されていない場合には、ソリューションエクスプローラーから「HelloWorld.c」をダブルクリックしてください。

▶ プログラムのビルドと実行

プログラムをビルドします。プログラムをビルドするには、[デバッグ(D)]メニューから[ソリューションのビルド(B)]を選択します。ソリューションウィンドウの右側の下側の「出力」というコンソールに、ビルド結果が表示されます。

正しくビルドできると、次のように表示されます。

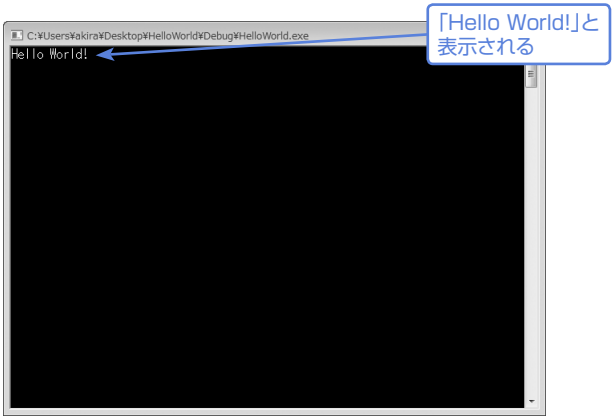
```
1>------ ビルド開始: プロジェクト: HelloWorld, 構成: Debug Win32 ------
1>  HelloWorld.c
1>  stdafx.cpp
1>  HelloWorld.vcxproj -> C:\Users\akira\Desktop\HelloWorld\Debug\HelloWorld.exe
========== ビルド: 1 正常終了、0 失敗、0 更新不要、0 スキップ ==========
```

プログラムコードの入力ミスなどがあると、「失敗」の左側にエラーの個数が表示されます。また、コードのどの行が間違っているのかなどのエラー情報も表示されます。たとえば、「printf」の行の末尾の「;」が抜けていると、次のように表示されます。

```
1>------ ビルド開始: プロジェクト: HelloWorld, 構成: Debug Win32 ------
1>  HelloWorld.c
1>c:\users\akira\desktop\helloworld\helloworld\helloworld.c(19): error C2146: 構文エラー
: ';' が、識別子 '_getch' の前に必要です。
========== ビルド: 0 正常終了、1 失敗、0 更新不要、0 スキップ ==========
```

エラーメッセージの行をダブルクリックすると、コード内の該当する行に自動的に移動して便利です。また、「F4」キーを押すと、次のエラーメッセージを選択し、コード内の該当する行に自動的に移動します。複数のエラーがあるときなどは「F4」キーを使った移動が非常に効率的です。

次に、プログラムを実行します。正常にビルドできたプログラムを実行するには、[デバッグ(D)]メニューから[デバッグ開始(S)]を選択します。「Hello World!」と表示されたコンソールウィンドウが表示されます。何かキーを押すと終了します。

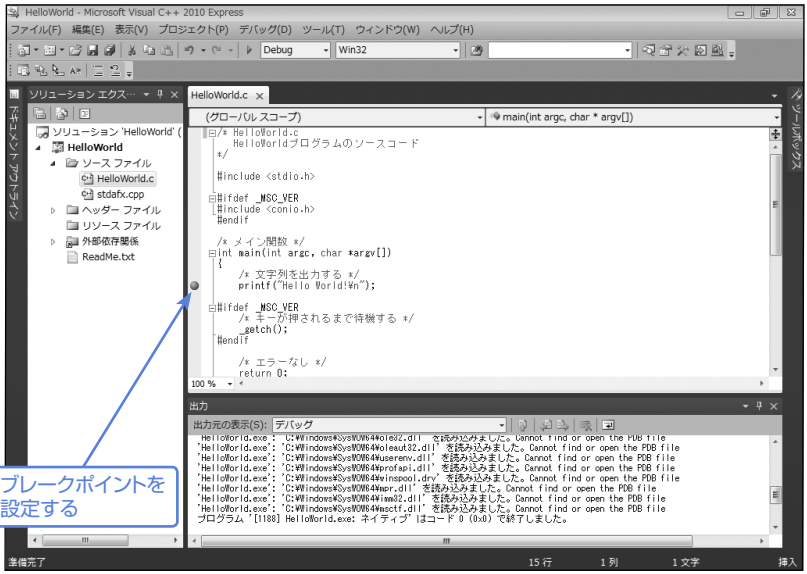


▶ デバッガの使い方

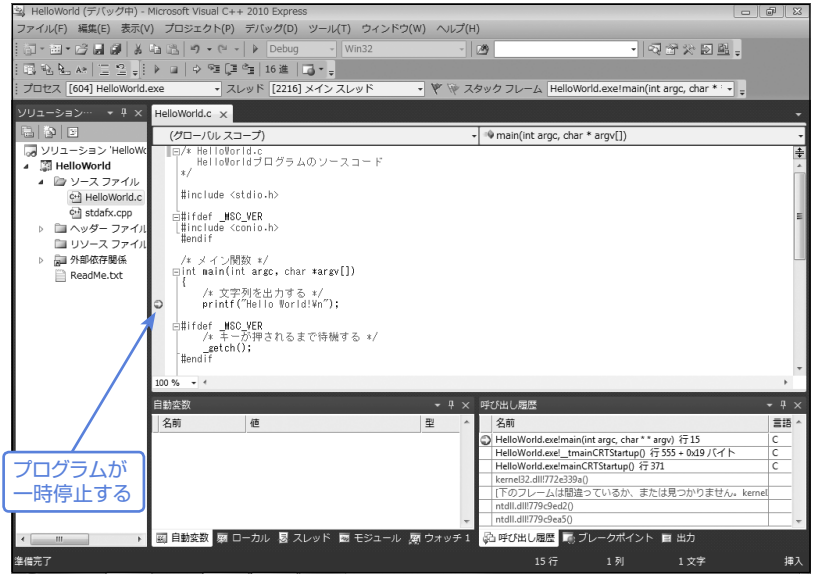
Visual C++はデバッガを内蔵しています。デバッガを使用すると、プログラムを任意の行で一時停止したり、1行ずつ実行したり、そのときどきの変数の内容を見ることなどができます。

特定の行でプログラムを一時停止するには、一時停止したい行にブレークポイントを設定します。ブレークポイントを設定するには、ブレークポイントを設定したい行にカーソルを移動し、[デバッグ(D)]メニューから[ブレークポイントの設定/解除(G)]を選択します。ブレークポイン

トが設定済みの行で再度、このコマンドを実行するとブレークポイントが解除されます。ブレークポイントが設定されると、その行の先頭に丸印が表示されます。なお、この丸印が表示される場所をクリックしてもブレークポイントの設定と解除を行うことができます。



ここで、「printf」関数を呼び出している行にブレークポイントを設定して、[デバッグ(D)]メニューから[デバッグ開始(S)]を選択してください。「printf」を呼び出す直前でプログラムが一時停止します。



デバッグ中は、Visual C++のウィンドウ内の表示もデバッグ用の表示になります。ウィンドウの右側の下側が左右に分割されます。左側のビューは変数の内容が表示されます。「自動変数」タブではプログラムの実行状態にあわせて自動的に関連する変数が表示されます。「ローカル」タブでは現在の関数内で有効な変数が表示されます。「スレッド」「モジュール」「ウォッチ1」は変数ではありません。「スレッド」は動作しているスレッドを表示し、「モジュール」はこのプログラムによって開かれているモジュールを表示します。「ウォッチ1」はウォッチ式を表示します。ウォッチ式は任意の変数や式を格納できます。

右側のビューは、「呼び出し履歴」「ブレークポイント」「出力」を切り替えて表示できるビューです。「呼び出し履歴」には、現在の関数が呼ばれるまでにどのような履歴で呼び出されているかが表示されます。このビューから呼び出し元の関数に移動することなどもできます。「ブレークポイント」はブレークポイントが一覧表示されます。

プログラムが一時停止している状態から動かすには、[デバッグ(D)]メニューのコマンドを実行します。各コマンドの機能は次の通りです。

コマンド	説明
[続行(C)]	次のブレークポイントまでプログラムを実行する
[デバッグの停止(E)]	プログラムを強制終了する
[ステップイン(I)]	呼び出そうとしている関数の中に入る
[ステップオーバー(O)]	ーステップ実行する
[ステップアウト(T)]	現在の関数の残りを実行して関数を抜ける

なお、これらのコマンドはツールバーに対応するボタンが用意されているので、そちらを使った方が便利です。

▶ファイルを追加する

このプログラムは1つのソースファイルで完結しますが、通常、ある程度の規模のプログラムでは関連する関数ごとにソースファイルを分け、複数のソースファイルで構成するのが一般的です。また、複数のソースファイルから使用される構造体の定義やプロトタイプの定義などはヘッダファイルに記述します。

Visual C++で、新しいソースファイルをプロジェクトに追加するには、次のようにします。

❶ [プロジェクト(P)]メニューから[新しい項目の追加(W)]を選択します。
❷ ダイアログボックスの左側のツリーから「コード」もしくは「Visual C++」を選択します。
❸ テンプレートから「C++ ファイル(.cpp)」を選択します。
❹ 名前にソースファイル名を入力します。このとき、拡張子の「.c」まで入力するとC言語用のソースファイルを作成できます。拡張子を省略すると、「.cpp」が使われ、C++用のソースファイルになります。
❺ [追加(A)]ボタンをクリックします。

ヘッダファイルを追加したいときも同様に操作し、テンプレートから「ヘッダーファイル(.h)」を選択します。拡張子は、省略すると、「.h」が使われます。

COLUMN 「_tmain」関数について

Visual C++のひな形から生成されるソースファイルでは、次のコードのように、「main」関数ではなく、「_tmain」関数という関数が実装されています。

```
int _tmain(int argc, _TCHAR* argv[])
{
        return 0;
}
```

Visual C++では、汎用テキストマッピングという機能が用意されています。「_tmain」関数は汎用テキストマッピングで提供される「main」関数で、Unicodeを使ったビルド設定では「wmain」関数となり、それ以外では「main」関数となる関数です。同様に、「TCHAR」型および「_TCHAR」型は、Unicodeを使ったビルド設定では「wchar_t」型となり、それ以外では「char」型となる型です。これにより、プログラムの国際対応が行いやすくなりますが、Windows専用のコードになってしまうという弱点もあります。

ビルド設定で、Unicodeを使用するかどうかは、次のように設定します。Visual C++ 2010 Expressでは、初期状態ではUnicodeを使用する設定になっています。

❶ ソリューションエクスプローラーからプロジェクトを選択します。
❷ [プロジェクト(P)]メニューから[プロパティ(P)]を選択します。ダイアログボックスが表示されます。
❸ 「構成」コンボボックスから「すべての構成」を選択します。
❹ 左側のツリーから「構成プロパティ」の「全般」を選択します。
❺ 「文字セット」から使用したい文字セットを選択します。
❻ [OK]ボタンをクリックします。

上記の手順で、❸はプロジェクトのすべての構成に対して設定したいときに選択します。特定の構成にのみ設定したい場合は、行わないでください。

Xcodeの操作方法

Xcodeでは、プログラムごとにプロジェクトを作成します。プロジェクトは複数のターゲットを持つことができ、1つのターゲットが1つのプログラムやライブラリをビルドします。関連するプログラムを1つのプロジェクトにターゲットとして登録することで、関連するプログラムのソースファイルも統合的に管理することができます。また、Xcode 4からは、関連するプロジェクトをまとめて管理することができるワークスペースも導入されました。

プログラムのソースファイルやヘッダファイルはプロジェクトに登録します。登録されたソースファイルは[ビルド]メニューから[ビルド]を選択するとコンパイルされ、プログラムがビルドされます。

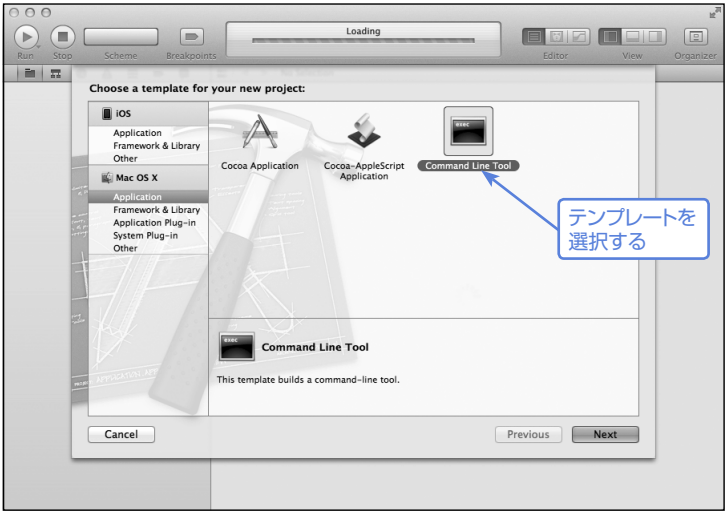
▶ プロジェクトの作成

次のように操作し、「HelloWorld」用のプロジェクトを作成します。

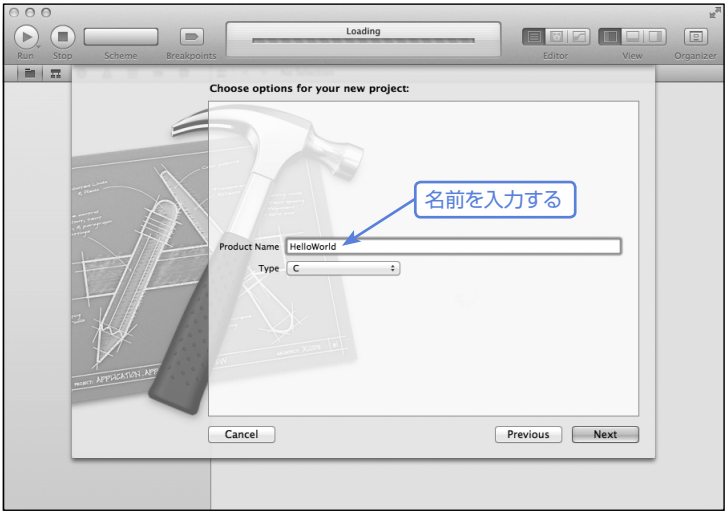
❶ [File]メニューから[New]→[New Project]を選択します。プロジェクトを作成するためのシートが表示されます。

❷ シートの左側のカテゴリから「Mac OS X」の「Application」を選択します。

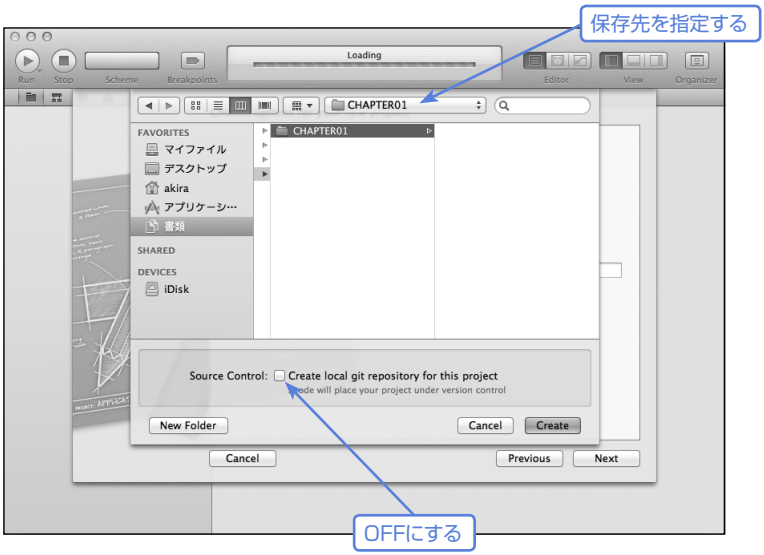
❸ シートの右側のテンプレートから「Command Line Tool」を選択し、[Next]ボタンをクリックします。



❹ プロジェクトのオプションを選択するシートが表示されるので、[Product Name]に「Hello World」と入力し、[Type]から「C」を選択して、[Next]ボタンをクリックします。

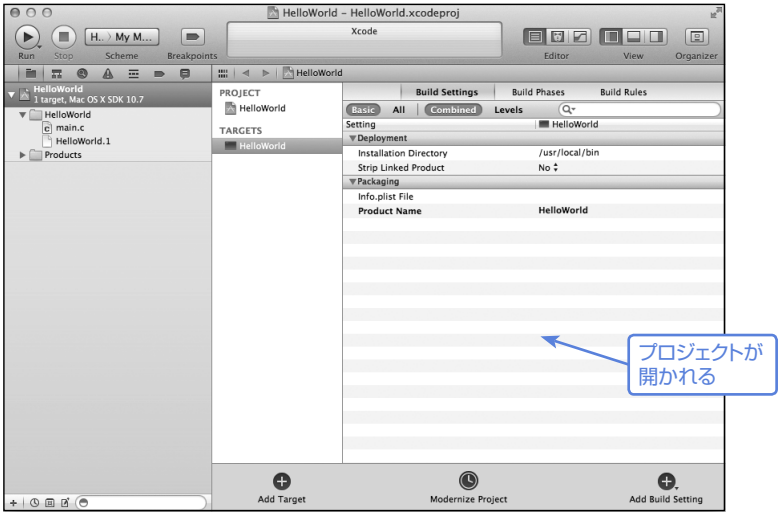


❺ 保存先を選択するシートが表示されるので、[Create local git repository for this project]をOFFにして、[Create]ボタンをクリックします。



❻ 選択した場所に「HelloWorld」フォルダが作成され、その中にプロジェクトファイルとひな形から作成されたソースファイルが保存されます。

❼ 保存されたプロジェクトが開かれます。



ワークスペースウインドウは、初期状態では左右に別れています。左側のエリアはナビゲータエリアと呼ばれる場所で、プロジェクトに登録されたファイルやグループが表示されます。ここには、コンパイルエラーなどの情報や、デバッグ時にはコールスタックなども表示されます。これらの表示は、ナビゲータエリアの上側に並んでいるボタンで切り替えることができます。右側のエリアはエディタエリアです。ナビゲータエリアで選択したファイルの内容が表示され、ここで

コードの編集も可能です。プロジェクトが開かれた直後の状態では、プロジェクトの設定画面が表示されています。

▶ コードの入力前に

コードを入力する前に、ひな形から作成されたファイルで必要ないファイルを削除します。次のように操作します。

❶ ナビゲータエリアから「HelloWorld.1」を選択します。

❷ [Edit]メニューから[Delete]を選択するか、キーボードの「delete」キーを押します。

❸ 削除方法を確認するダイアログボックスが表示されるので、[Delete]ボタンをクリックして、ファイルごと削除するようにします。

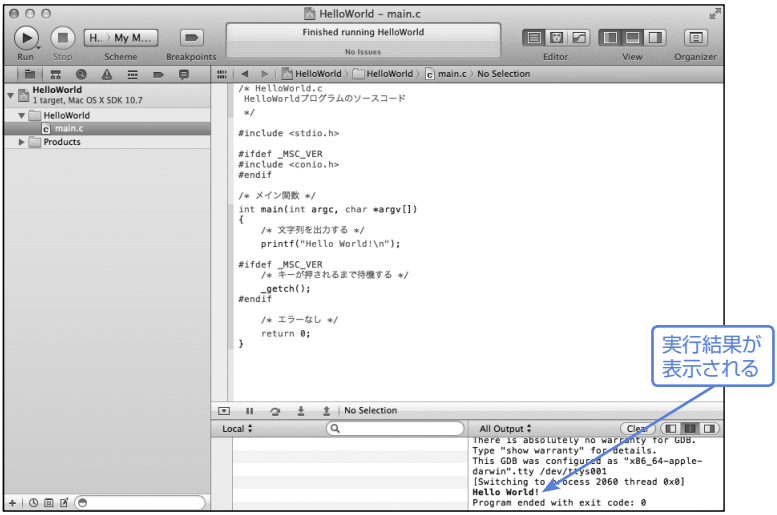
❹ ナビゲータエリアに登録されていた「HelloWorld.1」が削除されます。

▶ コードの入力

作成したプロジェクトにはあらかじめ、「main.c」が登録されているので、このファイルをそのまま使用します。ナビゲータエリアから「main.c」ファイルをクリックして選択します。エディタエリアにファイルが開かれるので、「HelloWorld.c」のコードを入力します。

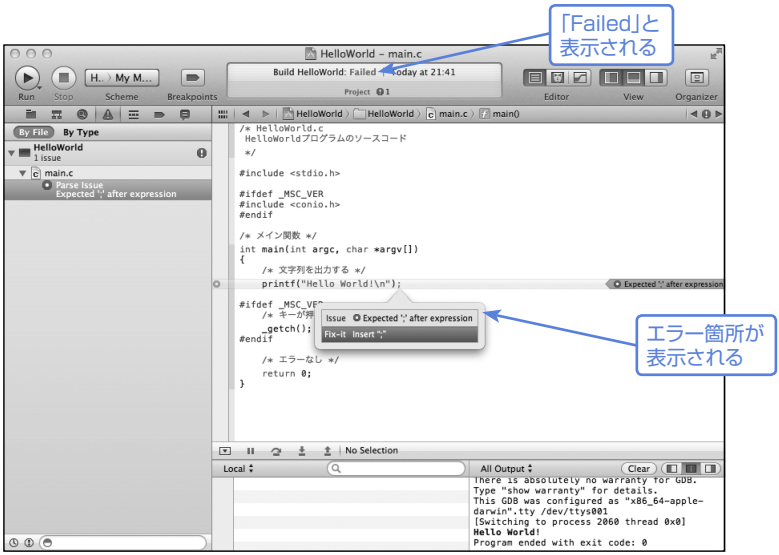
▶ プログラムのビルドと実行

プログラムをビルドして実行します。ワークスペースウインドウのツールバーから「Run」ボタンをクリックします。プログラムコードにミスがなく、ビルドに成功すると、そのままプログラムが実行され、実行結果がデバッグエリアのコンソールに表示されます。ここでは、「Hello World!」と表示されます。デバッグエリアは、エディタエリアの下側に表示されるエリアで、コンソールに出力されるときや、ブレークポイントでプログラムが一時停止するときに自動的に表示されます。



プログラムコードの入力ミスがあると、ワークスペースウインドウのツールバーのアクティビティビューア（液晶パネルのようなデザインのエリア）に「Failed」と表示され、エラーの個数が表示されます。ナビゲータエリアで「Issue」ボタンをクリックすると、具体的なエラー箇所がリストアッ

プされます。リストアップされた項目を選択すると、選択されたエラー箇所がエディタエリアに表示されます。たとえば、「printf」の行の末尾の「;」が抜けていると、次の図のように表示されます。



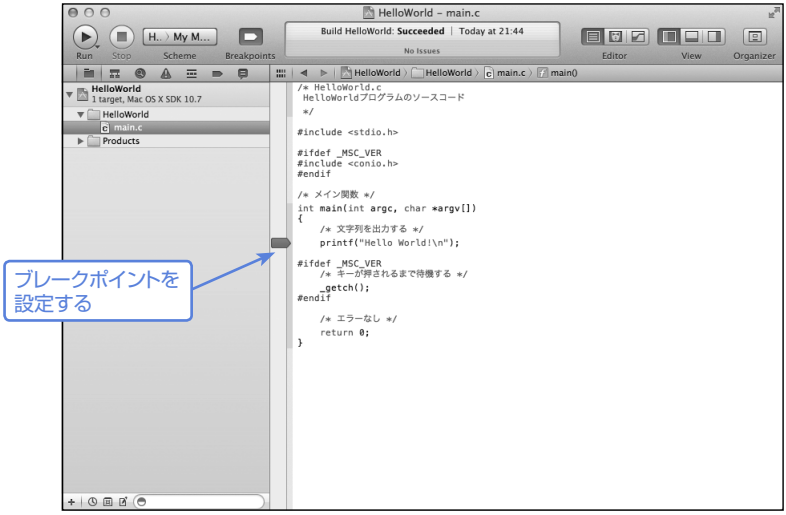
▶デバッガの使い方

XcodeはGDBのフロントエンドになる形でデバッガを内蔵しています。デバッガを使用すると、Visual C++の内蔵デバッガと同様に、プログラムを任意の行で一時停止したり、1行ずつ実行したり、そのときどきの変数の内容などを見ることができます。

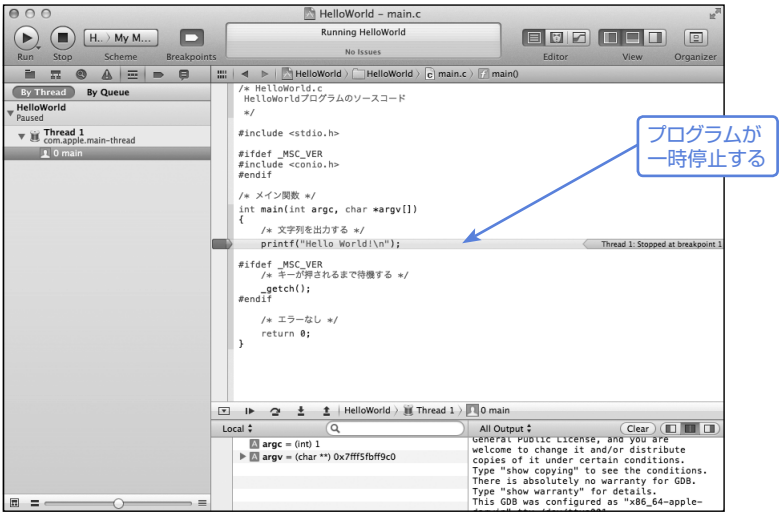
特定の行でプログラムを一時停止するには、一時停止したい行にブレークポイントを設定します。ブレークポインを設定するには、ブレークポイントを設定したい行にカーソルを移動し、[Product]メニューから[Debug]→[Add Breakpoint at Current Line]を選択します。ブレークポイントが設定されると、行の先頭に青い記号が表示されます。設定したブレークポイントを解除するには、ブレークポイントを設定した行にカーソルを移動し、[Product]メニューから[Debug]→[Remove Breakpoint at Current Line]を選択します。

なお、青い記号が表示された場所をクリックしても任意の行にブレークポイントを設定することができます。また、この青い記号を一度クリックすると、やや薄い表示になります。この状態のときはブレークポイントが一時的に無効となり、プログラムがその場所を実行するときに一時停止しません。

また、ブレークポイントの記号はドラッグして移動することもでき、この記号をドラッグしてブレークポイントが表示される領域の外にドロップしてもブレークポイントを解除することができます。マウス操作でのブレークポイントの設定の方が直感的に設定できるので、わかりやすいと思います。



ここで、「printf」関数を呼び出している行にブレークポイントを設定して、ワークスペースウインドウのツールバーの「Run」ボタンをクリックしてください。プログラムがデバッガ上で実行され、「printf」関数を呼び出す直前でプログラムが一時停止します。



変数の内容は、デバッグエリアのコンソールの左側のエリアに表示されます。また、ナビゲータエリアにはコールスタックが表示されます。現在、実行している関数がどのような履歴で呼び出されているかが表示され、実行中の関数を呼び出している関数などが表示されます。コールスタックで表示されている関数を選択すると、その関数のコードが表示されます。

プログラムが一時停止している状態から動かすには、デバッグエリアの上側に並んだボタンから「Continue」ボタン（左から2番目のボタン）をクリックします。［Product］メニューから

[Debug]→[Continue]を選択しても同様です。デバッグエリアの上側に並んだボタンの各機能は、次の通りです。

ボタン	説明
デバッグエリアの開閉ボタン(左端)	デバッグエリアを開いたり、閉じたりする
Continue	次のブレークポイントまでプログラムを実行する
Step Over	1ステップ実行する
Step Into	実行しようとしている関数の中に潜る
Step Out	実行中の関数の残りの処理を実行して、関数から抜ける

▶ファイルを追加する

複数のソースファイルで構成されるプログラムを作成するとき、Xcodeで、新しいソースファイルをプロジェクトに追加するには、次のようにします。

❶ [File]メニューから[New]→[New File]を選択します。

❷ テンプレートを選択するシートが表示されるので、シートの左側のカテゴリから「Mac OS X」の「C and C++」を選択し、テンプレートから「C File」を選択して、[Next]ボタンをクリックします。

❸ ファイル名と保存先を選択するシートが表示されるので、保存先を選択し、ファイル名を入力して、[Save]ボタンをクリックします。ここで入力する名前は、拡張子を含まないようにします。

❹ ソースファイルとヘッダファイルの両方が作成されて、プロジェクトに追加されます。

ソースファイルは作成せずに、ヘッダファイルだけを作成したいときは、❷のタイミングでテンプレートから「Header File」を選択します。

Ubuntu上での操作方法

Ubuntu上では、統合開発環境を使用せず、geditと端末を使用します。

▶フォルダの作成

本書では、1つのプログラムにつき、1つのフォルダを作成するようにします。ここでは「HelloWorld」というフォルダを作成してください。本書では、便宜上、フォルダをユーザのホームフォルダの直下に作成したものとして解説します。

▶コードの入力

コードを入力します。geditを起動し、「HelloWorld.c」のコードを入力します。入力が完了したら、作成した「HelloWorld」というフォルダに保存します。

▶プログラムのビルドと実行

作成したコードをコンパイルして、プログラムとしてビルドします。端末を起動し、次のように入力します。

❶ 次のように入力し、「HelloWorld」フォルダに移動します。「cd」コマンドは作業ディレクトリを変更するコマンドです。

```
cd ~/HelloWorld
```

❷ 次のように入力し、ソースファイルをコンパイルします。「GCC」はC言語のコンパイラです。「-o」オプションを指定すると、出力するファイル名を指定することができます。

```
gcc HelloWorld.c -o HelloWorld
```

ここではソースファイルは1つですが、複数のソースファイルがあるときは、スペース区切りでソースファイルを列挙します。

```
gcc HelloWorld.c HelloWorld2.c HelloWorld3.c -o HelloWorld
```

入力したコードに構文エラーなどがなく、ビルドに成功すると、「HelloWorld」フォルダに「HelloWorld」というプログラムが出力されます。プログラムを実行するには、端末で次のように入力します。

```
./HelloWorld
```

作成した「HelloWorld」が実行され、端末に次のように出力されます。

```
Hello World!
```

入力したプログラムコードに入力ミスがあると、コンパイルしたときにエラーメッセージが表示されます。たとえば、「printf」関数の行末の「;」が抜けていると、端末に次のように出力されます。

```
HelloWorld.c: In function ‘main’:
HelloWorld.c:23:5: error: expected ‘;’ before ‘return’
```

1行目は、「HelloWorld.c」の「main」関数の中を意味しています。2行目は、「HelloWorld.c」の23行目の「return」文の前に、「;」が必要だということを意味しています。

▶デバッガの使い方

Ubuntuには、デバッガ「GDB」も標準でインストールされています。GDBはコマンドラインから使用することができるデバッガです。Mac OS XのXcodeも内部ではGDBを呼び出しています。また、GDBにはフロントエンドとなるプログラムがいくつか公開されています。本書では、直接、端末からGDBを呼び出していますが、GUIでGDBを操作できるようになるフロントエンドプログラムを利用するのもよいと思います。

GDBを使用してデバッグを行うには、次のように操作します。

❶ デバッグシンボル付きでプログラムをビルドします。GCCでビルドするときに次のようにオプションを追加してビルドします。

```
gcc ソースファイル -o 出力ファイル名 -g3 -gdwarf-2
```

たとえば、「HelloWorld」であれば、次のように入力します。

```
gcc HelloWorld.c -o HelloWorld -g3 -gdwarf-2
```

❷ デバッグするプログラムを指定して、「GDB」を起動します。次のように入力します。

```
gdb プログラム名
```

たとえば、「HelloWorld」であれば、次のように入力します。

```
gdb HelloWorld
```

❸ ブレークポイントを設定します。ブレークポイントを設定するには、「break」コマンドを使用します。ブレークポイントは、関数名、もしくは、ソースファイルでの位置を指定します。

```
break 関数名
break ファイル名:行番号
```

たとえば、「HelloWorld.c」ファイルで「printf」関数を呼び出している行を指定する場合は次のように入力します。

```
break HelloWorld.c:15
```

❹ 「run」と入力し、プログラムをGDB上で実行します。

この手順に従って操作すると、「HelloWorld」が「GDB」上で実行され、「printf」関数を呼び出す直前一時停止し、端末に次のように出力されます。

```
Breakpoint 1, main (argc=1, argv=0xbffff8e4) at HelloWorld.c:15
15          printf("Hello World!\n");
(gdb)
```

「GDB」上で一時停止すると、Visual C++やXcodeと同様に、コールスタックを表示したり、変数の値を表示したり、1行ずつコードを実行したりすることができます。たとえば、「bt」と入力すると、次のようにコールスタックが表示されます。

```
(gdb) bt
#0  main (argc=1, argv=0xbffff8e4) at HelloWorld.c:15
```

また、「list」と入力すると、周辺のコードを表示することができます。

```
(gdb) list
10
11      /* メイン関数 */
12      int main(int argc, char *argv[])
13      {
14          /* 文字列を出力する */
15          printf("Hello World!\n");
16
17      #ifdef _MSC_VER
```

```
18          /* キーが押されるまで待機する */
19          _getch();
```

「continue」と入力すると、ブレークポイントが現れるまでプログラムを実行します。「Hello World」の場合は、他にはブレークポイントを設定していないので、プログラムを最後まで実行します。プログラムが正常に終了すると、次のように端末に出力されます。

```
Program exited normally.
```

「GDB」を終了するには、「quit」と入力します。

このように、「GDB」では、コマンドを入力してデバッグ作業を行います。「GDB」には非常に多くのコマンドが用意されていますが、頻繁に使用されるコマンドを挙げると、次のようになります。省略形が用意されているコマンドについては、省略形で入力することもできます。

コマンド	省略形	説明
backtrace	bt	コールスタックを表示する
break 関数名	b 関数名	指定した関数にブレークポイントを設定する
break ファイル名:行番号	b ファイル名:行番号	指定したソースファイルの行にブレークポイントを設定する
clear 関数名	—	関数に設定されたブレークポイントを削除する
clear ファイル名:行番号	—	指定したソースファイルの行からブレークポイントを削除する
continue	c	プログラムを再開する
finish	—	現在の関数が終わるまで実行する
info breakpoints	—	ブレークポイントの一覧を表示する
list	l	一時停止している行の周辺のコードを表示する
list 行番号	l 行番号	指定した行番号の周辺のコードを表示する
kill	—	プログラムを強制終了する
next	n	プログラムを1行実行する
print 変数名	p 変数名	変数の値を表示する。変数に式を指定した場合は、その結果を表示する
printf フォーマット文	—	変数の値を「printf」関数の構文に従って表示する
run	r	プログラムを起動する
step	s	プログラムを1行実行する。実行しようとしている行が関数の場合は、その関数の中に潜る

関連項目 ▶▶▶

- コメントについて …… p.62
- 配列について …… p.99
- プリプロセッサディレクティブについて …… p.129
- フォーマットを指定して文字列を出力する …… p.160

ライブラリについて

ライブラリとは

ライブラリとは、関数を提供するバイナリファイルです。複数のプログラムで共通して使用するであろう関数をライブラリにしておくと、各プログラムはライブラリを読み込むことで、これらの関数を利用できるようになります。プログラムからライブラリを読み込むことを「ライブラリとリンクする」と呼びます。C言語ではソースファイルをコンパイラがコンパイルして、ソースファイルごとにオブジェクトファイルを作成し、作成したオブジェクトファイルとライブラリをリンクして、1つのプログラムにします。

ライブラリには、静的ライブラリ(Static Library)と共有ライブラリ(Shared LibraryやDynamic Link Libraryなど)があります。静的ライブラリは、リンクさせたときにプログラムの中にそのまま組み込まれるライブラリです。一方、共有ライブラリは、リンクさせたときにはライブラリを参照するために必要な情報が書き込まれ、プログラムを実行するときに共有ライブラリをロードして動的にリンクされます。共有ライブラリの利点は、1つのオブジェクトコードを複数のプログラムで共有することができるので、プログラムのサイズを小さくしたり、共有ライブラリをアップデートすることで、プログラムは変更せずに機能の更新や不具合の修正を行うことができます。

ただし、欠点もあり、プログラムを実行するときに共有ライブラリが存在しなかったり、ロードできなかったりすると、その共有ライブラリを利用するプログラムはすべて実行できなくなってしまいます。静的ライブラリについてはこのような欠点はありませんが、プログラムの中にそのまま組み込まれるので、プログラムのサイズが大きくなってしまったり、ライブラリをアップデートしたときには、プログラムも再度、ビルドしなければ更新することはできません。

ライブラリには、開発環境に付属しているものがあります。付属しているライブラリを大きく分けると、次の2つに分かれます。

- 標準ライブラリ
- 開発対象専用のシステムライブラリ

開発では、これらのライブラリの他に、独自のライブラリを作成したり、開発するプログラムが利用する別のプログラムが提供する、API(「アプリケーション・プログラミング・インターフェイス」)を使用するためのライブラリなども使用します。

▶標準ライブラリについて

標準ライブラリは、規格で定義されているC言語の標準関数を提供しています。提供される機能はさまざまなものがあり、C言語を使ってプログラムを開発する上で非常に重要な役割を果たしています。標準ライブラリにより提供される関数には次のような種類があります。

- 文字の処理
- 文字列の処理
- 数値と文字列の変換処理

- ロケール処理
- 日時の処理
- 数学処理
- 複素数の処理
- 入出力処理
- エラー処理
- シグナル処理
- 可変引数処理
- その他、ユーティリティ的な処理

▶開発対象専用のシステムライブラリについて

開発対象専用のシステムライブラリは、システムコールを提供するライブラリです。プログラムを実行するシステム特有の機能を呼び出すための関数をシステムコールと呼びます。システムコールは各システム特有のもので、あるシステムのシステムコールは別のシステム上では使用できません。一般的に、C言語で開発するプログラムは標準ライブラリの処理とシステムコール、各プログラム特有の関数の組み合わせで開発されるものなので、あるシステム用に開発されたプログラムは別のシステムでは実行できません。

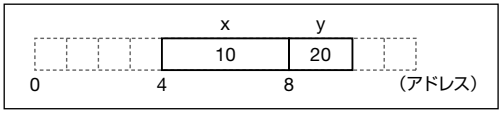
ただ、「POSIX.1」のように、規格化されているシステムコールもあります。また、各ベンダーや複数のシステムに対応したプログラムが提供するAPIなどでは、システムが異なってもソースコードのレベルでは互換性が保たれるようになっているものもあります。

メモリ領域について

メモリ領域と変数について

C言語には変数と呼ばれる値を入れておくことできる場所があります。この変数はメモリ領域と呼ばれる場所に作成されます。このメモリ領域がどの程度の大きさがあるかは、プログラムを実行するハードウェアやOSの設定によって変わってきます。

たとえば、変数「x」に10、変数「y」に20という値が入っているとすると、次の図のようなイメージになります。



メモリ領域では「バイト」という単位で大きさやメモリ領域内での位置を示します。この図では点線で囲まれた1つの正方形が1バイトを表しています。変数「x」は4バイトの大きさがあり、メモリ領域内で4の位置にあるといえます。変数「y」は2バイトの大きさがあり、メモリ領域内で8の位置にあるといえます。このメモリ領域内でどこにあるかという、位置のことを「アドレス」と呼びます。

この図のように、変数によって、使用するメモリサイズが異なります。なぜ、異なるのかというと、メモリサイズを大きく使うほど、よりたくさんの情報を入れることができるからです。C言語では、変数に入れる情報の量や種類によって、大きな変数や小さな変数を使い分けます。種類の違いを「型」と呼びます。「型」には整数や浮動小数点数などがあり、変数を定義するときに、どの型を使用するか指定します。変数の定義方法については、《**変数について**》(p.68)を参照してください。

▶バイトとビットについて

メモリ領域での単位となる「バイト」は、さらに「ビット」で構成されています。「ビット」は「ON」と「OFF」という2つの状態を持ちます。「ON」が「1」であり、「OFF」が「0」となります。PCで使われる一般的なCPUでは、8ビットで1バイトとなります。1バイトが8ビットではないというCPUも存在しますが、本書では8ビットが1バイトとなる動作環境を対象としています。

ビットは「0」か「1」のどちらかにしかならないので、2進数を使って表現することができ、1バイトの最小値と最大値は次のように表現できます。

```
最小値 : 00000000 = 0
最大値 : 11111111 = 255
```

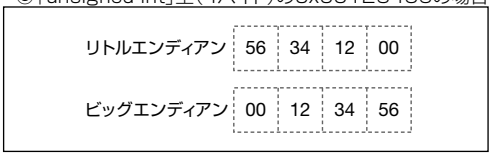
つまり、負の値を考えなければ、1バイトは0以上255以下の値を表現できることになります。負の値も考慮すると、-128以上127以下の値となります。この範囲外の値を表現するためには、複数のバイトを使って表現する必要があります。そのため、大きな変数や小さな変数が生まれます。

▶ エンディアンについて

複数バイトで構成される変数がメモリに配置されるときやファイルなどに書き出されるときに、どの順番でバイトを並べるかというルールを「エンディアン」と呼びます。「エンディアン」はCPUやOSなど、動作環境によって変わってきます。上位バイトから順に配置する方法を「ビッグエンディアン」と呼び、下位バイトから配置する方法を「リトルエンディアン」と呼びます。

一般的にx86互換のCPUでは「リトルエンディアン」が使われ、Power PC系のCPUでは「ビッグエンディアン」が使われています。iPhoneやiPadでは「リトルエンディアン」が使われています。

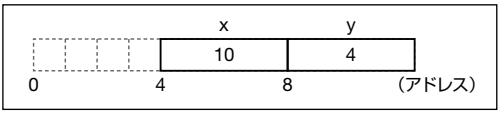
◉「unsigned int」型(4バイト)の0x00123456の場合



ポインタについて

C言語での重要な概念に「ポインタ」と呼ばれるものがあります。「ポインタ」は英語で書くと「Pointer」です。この単語は「指し示す物」という意味です。C言語でのポインタも同様で、ある変数のことを指し示すことです。変数を指し示すために必要なものは「アドレス」です。言い換えれば、ポインタは指し示す変数のアドレスのことであり、ポインタとなる変数には別の変数のアドレスが格納されます。コードでどのようにポインタを記述するかについては、《**演算子について**》(p.81)を参照してください。

たとえば、変数「x」に「10」という値が入っており、変数「y」が変数「x」のポインタだとすると、次の図のようなイメージになります。



変数「x」のアドレスは「4」なので、変数「y」には「4」が入っています。

▶ 「NULL」について

ポインタで、「何も指し示していない状態」を表現するために「NULL」というものがあります。「NULL」は数値の「0」と等価です。値が「NULL」になっているポインタは無効なポインタと考えます。

メモリ領域の種類

メモリ領域には、次のような種類があります。

- プログラム領域
- 静的領域
- スタック領域
- ヒープ領域

▶プログラム領域について

プログラム領域は、実行するプログラムが読み込まれる領域です。

▶スタック領域について

スタック領域は、自動変数が格納される領域です。自動変数とは、その変数が定義された関数やスコープ(「{」と「}」で囲まれた範囲)を抜けると自動的に解放される変数です。

スタックは「後入れ先出し(LIFO=Last In First Out)」のデータ構造で、後から入れた物が先に取り出されるというデータ構造です。たとえば、次のようなコードがあるとします。

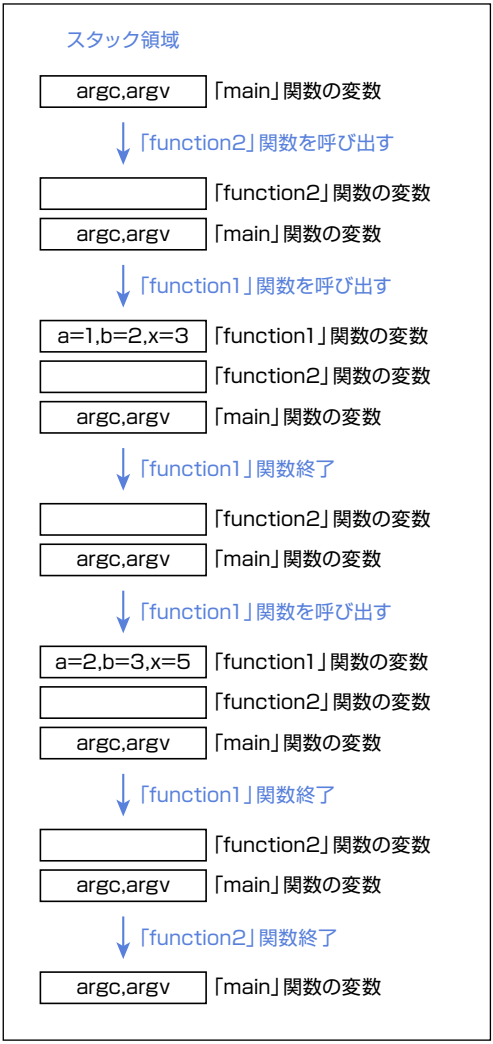
```
void function1(int a, int b)
{
    int x;
    x = a + b;
}

void function2()
{
        function1(1, 2);
        function1(2, 3);
}

int main(int argc, char *argv[])
{
    function2();
    return 0;
}
```

このコードは、「main」関数が「function2」関数を呼び出し、「function2」関数が「function1」関数を呼び出すという流れになっています。このときのスタック領域の変化は、次ページのようなイメージになります。

なお、スタック領域は一般的に大きなサイズの変数を扱うことはできません。スタック領域はヒープ領域に比較して非常に小さい場合が多く、数MBなどのまとまったデータを扱うにはヒープ領域に確保する必要があります。



▶ヒープ領域について

ヒープ領域は、ある程度大きなサイズの変数も扱うことができるメモリ領域です。「malloc」関数などを使い、必要な容量だけメモリブロックを確保して使用します。スタック領域に確保される自動変数とは異なり、メモリブロックを確保した関数が終了しても、確保したメモリブロックは解放されません。ヒープ領域から確保したメモリブロックは必要なくなった時点で、「free」関数など、メモリブロックの解放関数を使用して解放する必要があります。ヒープ領域からメモリブロックを確保する方法の詳細については、《**メモリブロックを確保する**》(p.238)を参照してください。