第 1 章　C 言語開発環境の構築

1.1　C 言語とは

プログラミング言語「C」は、1972 年にベル研究所のデニス・M・リッチーが開発したプログラミング言語です。この名前は、同じくベル研究所で開発されたプログラミング言語「B」に由来します。正式名称は上記のとおり「C」ですが、若干わかりにくいので本書では便宜上「C 言語」と呼ぶことにします。もともと C 言語は、UNIX というオペレーティングシステムの移植性を高めるために作られました。しかし現在では、その利便性からさまざまな用途に活用されています。

世の中には、C 言語以外にもさまざまなプログラミング言語が存在します。例えば、C++、Java、Python、PHP などです。その中から、なぜ我々は C 言語を学ぼうとしているのか、その動機についても軽く触れておきましょう。C 言語は非常に軽量で、単純で、さまざまな領域で活用可能な言語です。これらの特徴は、アルゴリズムの学習や、論理的思考能力の習得に適していると言えます。しかし最も注目すべきは、多くのプログラミング言語に影響を与えてきたという点です。上述した 4 つのプログラミング言語はいずれも C 言語から派生したものであり、その他にも C 言語から派生した言語は多数存在します。つまり、C 言語で基本的な文法を理解しておけば、他の多くの言語の習得がスムーズになるということです。C 言語を学んで損をすることはありませんので、ためらわず、学習に励みましょう。

1.2　開発に必要なもの

C 言語でプログラミングをするには、次のものが必要になります。

コンピュータ

当然ですが、大前提としてコンピュータが必要です。本書では、基本的に OS（オペレーティングシステム）として、Windows が搭載されたコンピュータを対象とします。

エディタ

プログラムのソースコードを書くためのソフトウェア（エディタ）も当然必要です。Windows では標準でメモ帳（Notepad）などがインストールされていますが、機能が少なく扱いにくいので、「秀丸」と呼ばれる高機能エディタのインストールを推奨します。

コンパイラ

エディタで書いたソースコードは、コンパイラ（翻訳機）を用いて、コンピュータが理解できる機械語にコンパイル（翻訳）する必要があります。本書では Cygwin という UNIX ライクな環境と、gcc というコンパイラをインストールします。

1.3　エディタ（秀丸）のインストール

それでは早速、秀丸エディタのインストールから始めましょう。以下のリンクにある「通常の最新版」から、exe ファイルをダウンロードして実行します。

（秀まるおのホームページ）

http://hide.maruo.co.jp/software/hidemaru.html

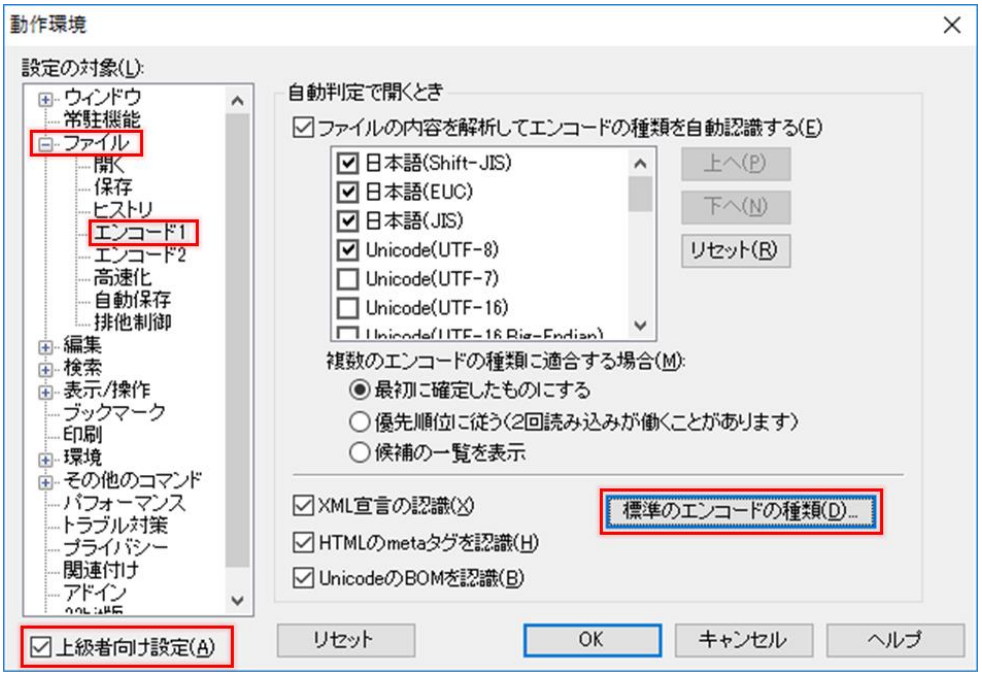
※2016 年 9 月 28 日時点での最新バージョンは、「hm864_signed.exe」です。

秀丸のインストールは、基本的に「次へ」ボタンを押していけば完了します。

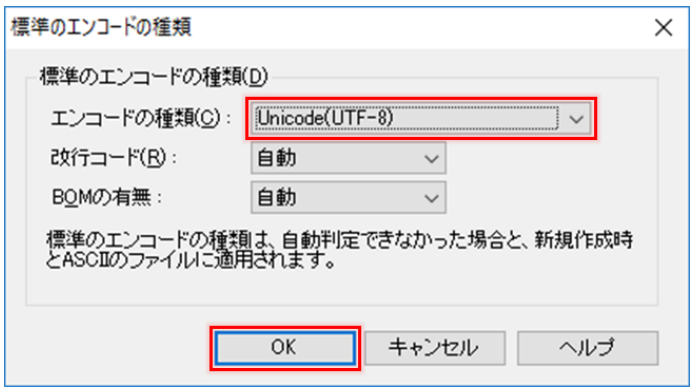
文字のエンコード設定

インストールが完了したら、標準の文字のエンコードを設定しておきましょう。文字のエンコードとは、文字をコンピュータ上で表現する方式のことで、保存した環境と開く環境でエンコードが異なると、文字化けを起こします。秀丸では、標準の文字のエンコードは「日本語(Shift-JIS)」となっていますが、これを「Unicode(UTF-8)」に変更します。「UTF-8」は国際標準のエンコードであり、あらゆる国の言語を文字化けせず表示でき、ほぼ全てのコンピュータで扱うことができます。

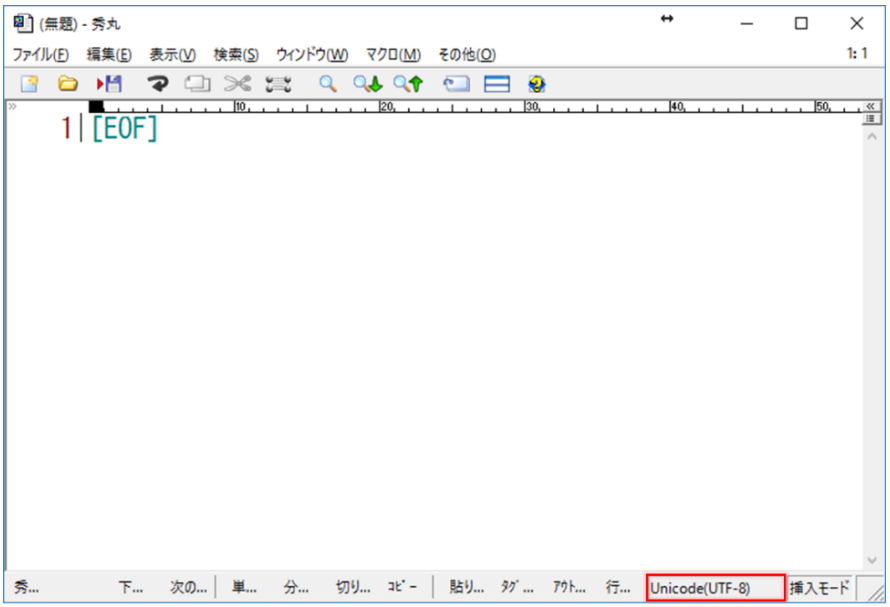
秀丸を立ち上げたら、メニューバーから［その他］→［動作環境］と進みます。動作環境ウィンドウが出たら、左下の［上級者向け設定］にチェックを入れ、［ファイル］→［エンコード 1］→［標準のエンコードの種類］と進みます。



標準のエンコードの種類が「日本語(Shift-JIS)」になっていたら、「Unicode(UTF-8)」に変更して、［OK］を押してウィンドウを閉じます。



一度秀丸を閉じて、再び開いてみましょう。右下のステータスバーが、「Unicode(UTF-8)」となっていたら成功です。また、右下のエンコード名の領域をクリックすることで、個別にエンコードを変更することもできます。もし右下にエンコードの種類が表示されていない場合は、メニューバーの［表示］から、［ステータスバー］を選択して下さい。



秀丸の料金

本来秀丸はシェアウェアであり、4,320 円（2016 年 9 月 28 日時点）します。しかし、金銭的に難儀している学生であれば、秀丸を無料で使うことができます。

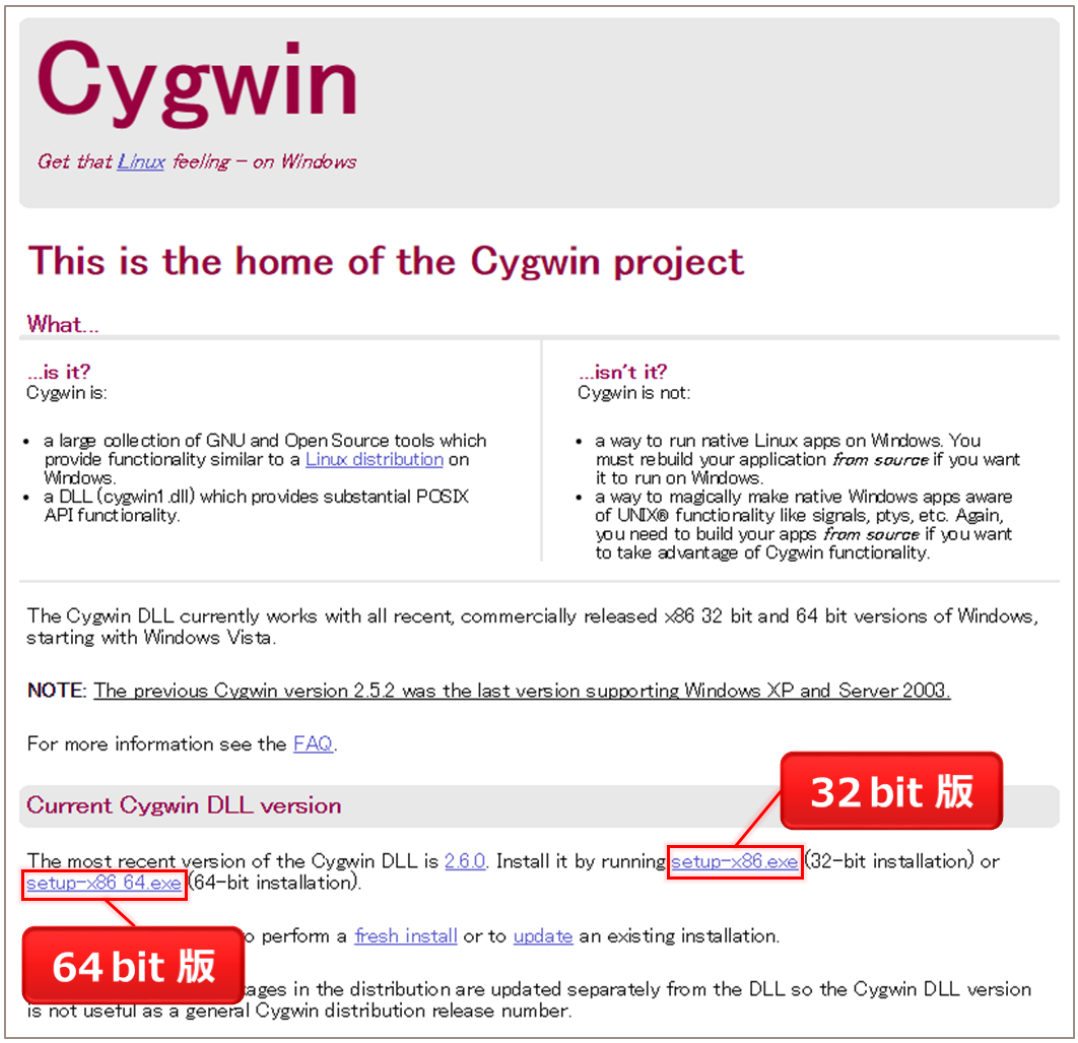
秀まるおのホームページから、［サポート］→［秀丸エディタフリー制度］→［アカデミックフリー個人］と進み、学校のメールアドレス（ac.jp で終わるもの）を用いて登録申請を行って下さい。もし秀丸のことが気に入って、大学卒業後も使っていきたいとなった場合は、お金を払いましょう。

1.4　コンパイラ（Cygwin, gcc）のインストール

次は、コンパイラ（翻訳機）のインストールです。1.2 節で述べたとおり、プログラムのソースコードは、コンピュータが理解できる機械語にコンパイル（翻訳）する必要があります。本書では、gcc（GNU Compiler Collection）と呼ばれる、GNU プロジェクトが開発・公開しているコンパイラを利用します。また、gcc は UNIX 系 OS との相性が良いので、Windows 上で UNIX ライクな環境を簡単に構築できる Cygwin を、同時にインストールします。以下のサイトより、最新版の Cygwin インストーラをダウンロードして実行しましょう（インストーラには 32 bit 版と 64 bit 版があります）。

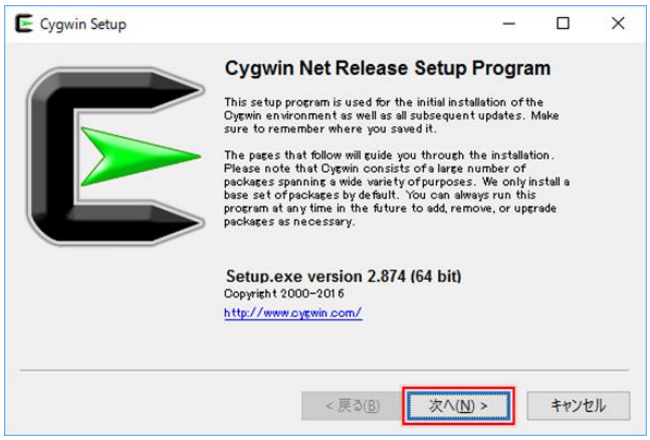
（Cygwin 公式ページ）

https://www.cygwin.com/

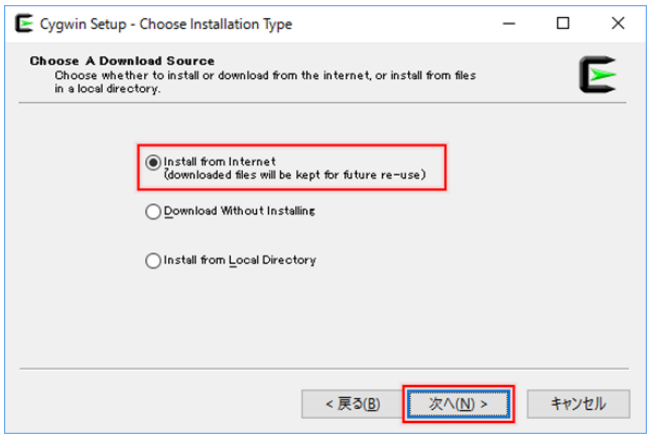


※2016 年 9 月 28 日時点での最新バージョンは、「2.6.0」です。

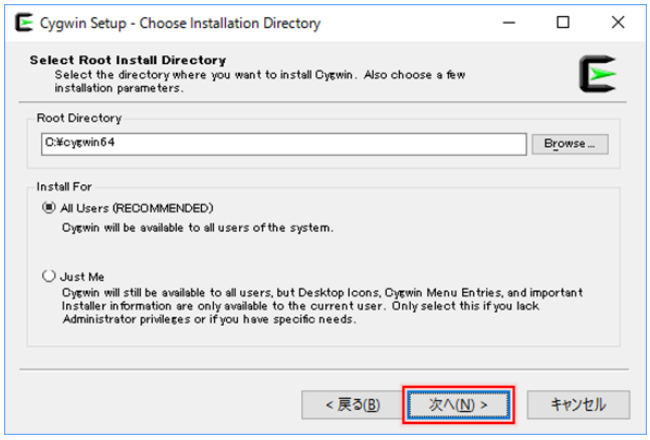
① インストーラを立ち上げると、次のような画面になります。[次へ] ボタンを押して進みましょう。



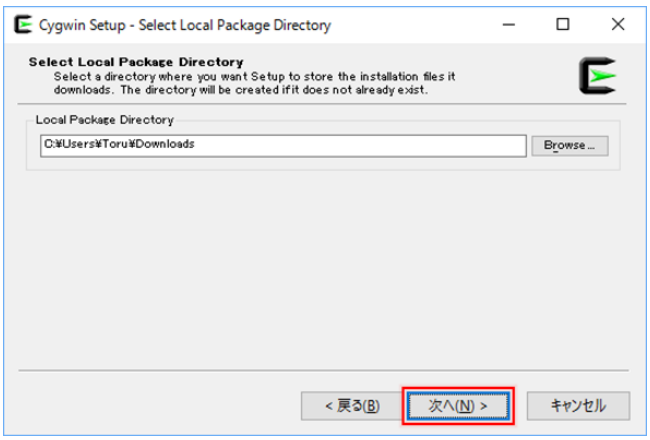
② データのダウンロード方法を選択します。[Install from Internet] がチェックされていることを確認し、[次へ] を押します。



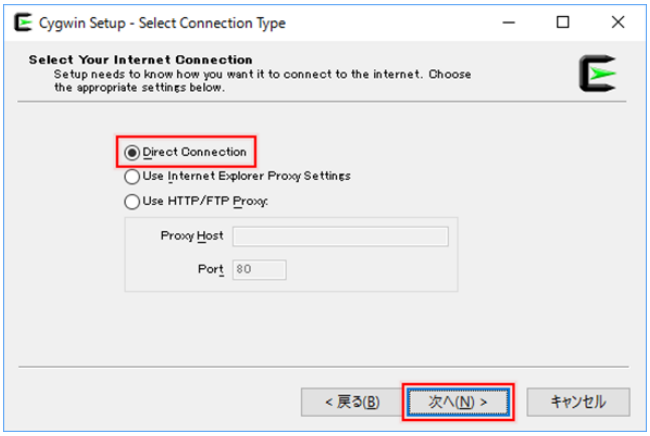
③ Cygwin のインストールフォルダ（Root Directory）を選択します。特にこだわりが無ければ、そのまま [次へ] を押します。



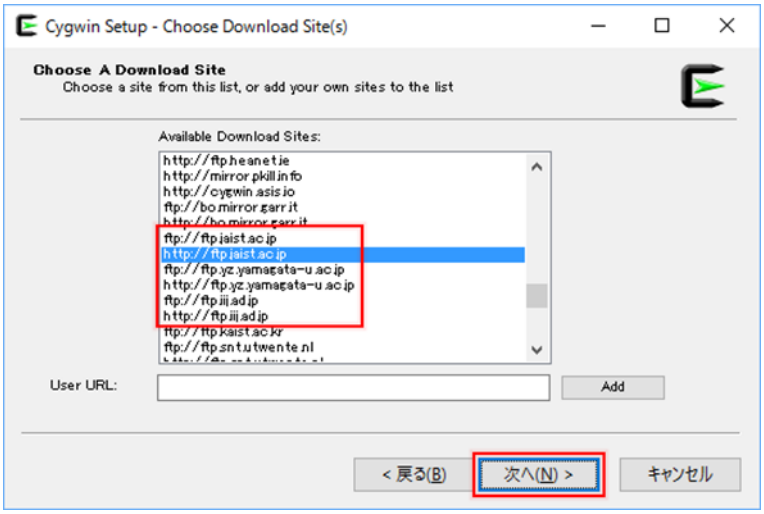
④ 一時的にインストールファイルを保存するフォルダを選択します。特にこだわりが無ければ、そのまま［次へ］を押します。ここで指定したフォルダに生成されたファイルは、Cygwin のインストール後に削除しても大丈夫です。



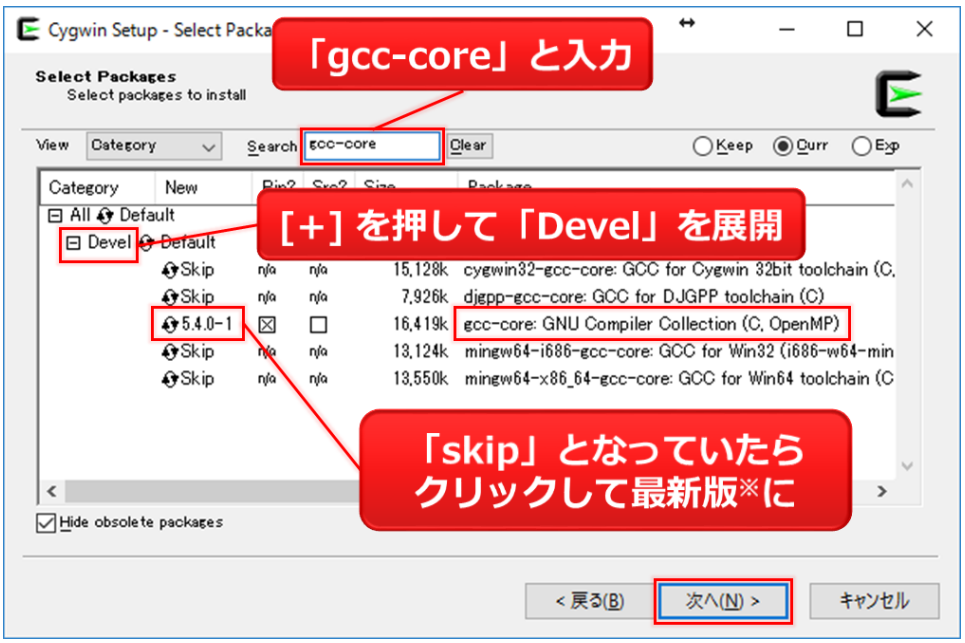
⑤ インターネット接続の方式を選択します。特にこだわりが無ければ、そのまま［次へ］を押します。



⑥ ダウンロードサーバを選択します。安定したダウンロードのため、「.jp」で終わるドメインを探し、選択します。「ftp」と「http」はどちらでも構いません。

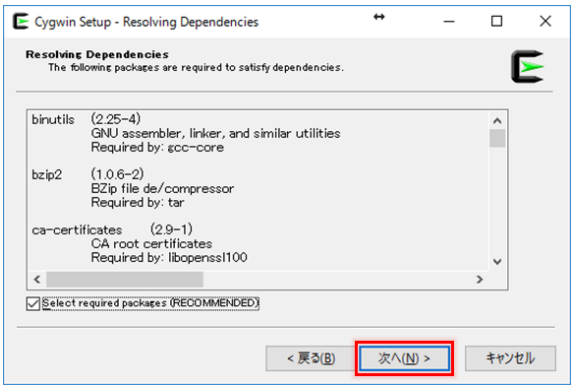


⑦ インストールするパッケージを選択します。「gcc」の核の部分だけインストールすれば良いので、[Search] と書かれた領域に「gcc-core」と入力し、フィルタリングします。[Devel] を展開して「gcc-core: GNU Compiler Collention (C, OpenMP)」を探し、[Skip] となっていたらクリックして最新版をインストールするように変更します。

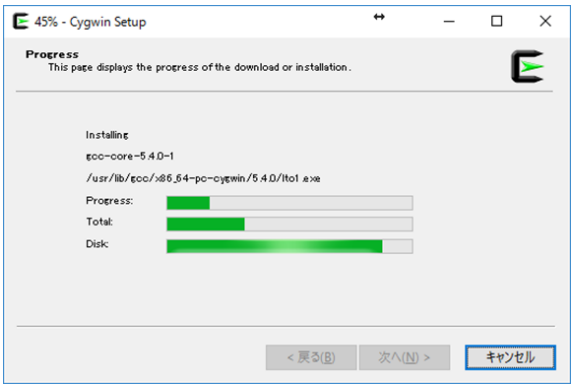


※2016 年 09 月 28 日時点での最新バージョンは 5.4.0-1 です。

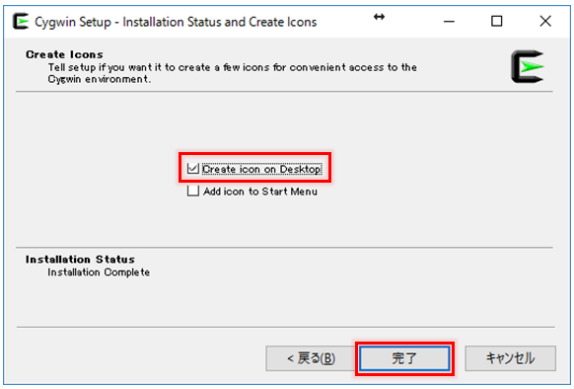
⑧ 依存性の問題を解消するため、追加で必要なパッケージを自動的に選択してくれます。そのまま［次へ］を押しましょう。



⑨ ダウンロードとインストールが始まるので、暫く待ちます。



⑩ インストールが終了したら、［完了］ボタンを押して終わります。下の図では、デスクトップにアイコンを作るオプションを選択しています。



以上で Cygwin および gcc のインストールは終了です。Cygwin のフォントを変更したい場合は、Cygwin を起動して画面上で右クリックし、[Options...] → [Text] → [(Font)Select...] と進んで下さい。

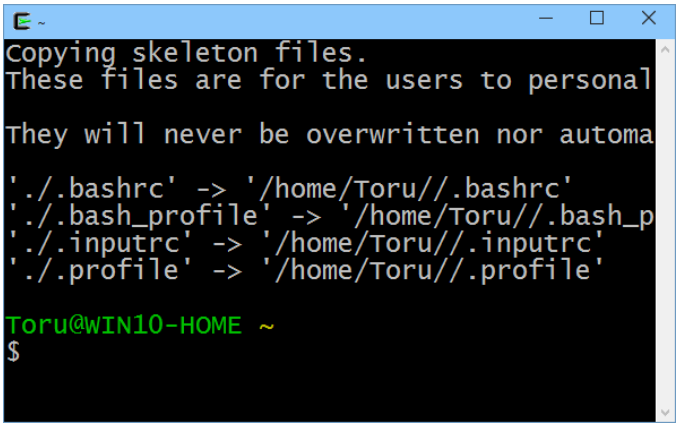
1.5　UNIX コマンドによる操作

Cygwin では、多くの UNIX コマンドが利用可能となっています。その中でも、今後よく使うであろうものを、以下にリストアップします。

コマンド名	機能
pwd	現在のディレクトリを表示
ls	現在のディレクトリのファイル一覧を表示
mkdir [dir_name]	新規ディレクトリの作成
cd [dir_name]	指定したディレクトリに移動
cd ..	一つ上のディレクトリに移動
touch [file_name]	新規ファイルの作成
rm [file_name]	指定ファイルの削除
cp [fileA] [fileB]	ファイル A からファイル B にコピー
cygstart [file_name]	ファイルを開く（ダブルクリックと同じ処理）
cygstart .	現在のディレクトリをエクスプローラで開く
gcc [file_name]	C 言語のファイルをコンパイル
./[file_name]	コンパイルされたファイルを実行
exit	Cygwin を終了

※「ディレクトリ」は、Windows の「フォルダ」に相当します。

一度に全て覚えるのは難しいと思うので、少しずつ使いながら覚えていきましょう。Cygwin を起動すると、以下のような画面が立ち上がります。これから、ここにさまざまなコマンドを打ち込んでいきます。



現在位置の確認

まずは現在自分がいる位置（ディレクトリ）を確認してみましょう。以下のコマンドを打ち込んで下さい。

```
$ pwd
```

すると、「/home/user_name」のように、現在の位置が表示されます。pwd は「print working directory」を意味します。

作業ディレクトリの作成

これからプログラムを作るための、作業ディレクトリ（作業フォルダ）を作成しましょう。以下のように「mkdir」(make directory)コマンドを書き、半角スペースを空けてディレクトリ名を記述します。

```
$ mkdir work
```

ここでは work という名前のディレクトリを作成しました。何も表示されなければ、成功です。UNIX では多くの場合、コマンドの実行に成功すると沈黙を守ります。余計な出力は開発者にとって邪魔なだけであることを主張した、UNIX 哲学の一つです。ただ、今回は初めての作業なので、「ls」(list)コマンド、もしくは「cygstart .」コマンドでディレクトリが作成されたことを確認しておきましょう。

```
$ ls
$ cygstart .
```

「ls」は、現在のディレクトリに含まれるファイルやディレクトリの一覧を表示するコマンドです。「work」ディレクトリが表示されたら成功です。「cygstart .」は cygwin 独自のコマンドで、現在のディレクトリを Windows のエクスプローラで開いてくれます。エクスプローラ上でも、「word」ディレクトリ（フォルダ）が作成されていることを確認できます。

ディレクトリの移動

それでは、作成したディレクトリの中に移動しましょう。以下のように「cd」(change directory）コマンドで、移動先を指定します。

```
$ cd work
```

緑色で表示されたユーザ名の隣に、黄色で「~/work」のように表示されていれば、移動は成功です。「ls」コマンドでフォルダの中に何も入っていないことを確認しておきましょう。

ソースファイルの作成

次は、実際に C 言語のソースファイル（*.c)を作成します。エクスプローラ上や秀丸上で作ることも可能ですが、コマンドで作る方法も覚えておきましょう。以下のように、「touch」コマンドを用います。

```
$ touch hello.c
```

「ls」コマンドを打って、正しく作成されたことを確認しておきましょう。また、以下のように「cygstart」でファイル名を指定すれば、既定のプログラムを用いてファイルを開くことができます。

```
$ cygstart hello.c
```

既定のプログラムがメモ帳などである場合は、エクスプローラ上で C 言語ソースファイルを右クリックし、「プロパティ」もしくは「プログラムから開く」から既定のプログラムを秀丸に変更しておくことをおすすめします。

1.6　文字の出力（Hello, world!）

それでは、C 言語のソースコードを書いていきましょう。まずは恒例の「hello world」です。以下のソースコードを「hello.c」に書いて、保存して下さい。

```c
#include <stdio.h>

int main(void) {
    printf("Hello, world!\n");
    return 0;
}
```

一行ずつソースコードの意味を見ていきましょう。今はまだ、全てを完璧に理解する必要はありません。

1 行目：「stdio.h」というヘッダファイルを読み込んで（インクルードして）います。「stdio」は「standard input/output」の略であり、「stdio.h」には入出力に関するさまざまな命令が含まれています。C 言語では、どんなプログラムを書く場合も、通常はこの行から始まります。

2 行目：ソースコードを読みやすくするための、空行です。

3 行目：main 関数と呼ばれる関数（一連の処理のまとまり）で、C 言語のソースファイルには必ず 1 つだけ存在し、プログラムはここから始まります。波括弧 { } で囲まれた領域が、main 関数の中身になります。「main」の文字の左右にある「int」と「void」についての詳細は、第 5 章「関数」にて説明します。

4 行目：文字列を画面に表示（プリント）する、printf 関数を呼び出しています。printf 関数ではこのように、表示したい文字列をダブルクォーテーション（“”）で囲います。文字列の最後にある「\n」は、改行を意味します。日本語キーボードで「\」を入力する際は、プログラム上で同じ意味を示す「¥」を入力して下さい。また、右端のセミコロン（;）は行の終わりを意味します。

5 行目：main 関数（およびプログラム）が正常に終了したことを報告する行です。「return」は関数の終了を、「0」は正常終了を意味します。

ソースファイルのコンパイル

ソースコードの記入・保存ができたら、Cygwin の画面に戻って下さい。C 言語のソースコードを、コンピュータが理解できる機械語（0 と 1 のみで構成された言語）にコンパイル（翻訳）しましょう。以下のように、「gcc」コマンドでコンパイルしたいファイルを指定します。

```
$ gcc hello.c
```

何も表示されなければ、コンパイルは成功です。「ls」コマンドを実行してみましょう。「a.exe」という名前の実行ファイルができあがっているはずです。エラーメッセージが出る場合は、エラー内容をよく読んで、ソースファイルを修正してから再挑戦して下さい。

実行ファイル名を「a.exe」ではなく、「hello.exe」など固有の名前を付けたい場合は、「-o」オプションを利用して以下のように書きます。

```
$ gcc -o hello hello.c
```

プログラムの実行

生成された実行ファイル（exe ファイル）を実行してみましょう。以下のように、「./」に続けて実行ファイル名を打ち込みます。

```
$ ./a.exe
```

Cygwin 上に「Hello, world!」と表示されたら成功です。「.exe」の部分は省略して、以下のように書くこともできます。

```
$ ./a
```

実行ファイル名が「hello.exe」の場合は、以下のように記述します。

```
$ ./hello.exe
$ ./hello
```

以上が、ソースコードの作成から実行までの流れになります。

1.7　プログラム実行までの流れのまとめ

Cygwin を起動してからプログラムを実行するまでの流れをおさらいしてみましょう。「pwd」コマンドや「ls」コマンドによる確認操作は省略します。

1. 作業ディレクトリへの移動

Cygwin を起動したら、まずは「cd」コマンドで作業ディレクトリに移動します。

```
$ cd work
```

新しく作業ディレクトリを作る場合は、「mkdir」コマンドを実行します。

2. ソースファイルの作成

作業ディレクトリに移動したら、「touch」コマンドでソースファイルを作成します。

```
$ touch hello.c
```

作成したファイルを規定のプログラムで開くには、「cygstart」コマンドを用います。

```
$ cygstart hello.c
```

3. ソースファイルのコンパイル

秀丸などのエディタでソースファイルの編集が完了したら、「gcc」コマンドでコンパイルします。名前を指定しなかった場合は、「a.exe」という実行ファイルが生成されます。

```
$ gcc hello.c
```

固有の名前（hello）を付けたい場合は、以下のように「-o」オプションを用います。

```
$ gcc -o hello hello.c
```

4. プログラムの実行

コンパイルに成功したら、「./」に続けて実行ファイル名を打ち込み、実行します。（「.exe」は省略可）

```
$ ./a
$ ./hello
```

1.8　コメントとインデント

ソースコードには、コメントを含めることができます。コメントとは、プログラムの実行とは何の関係もない、プログラムによるメモ書きのようなものです。以下のように、「/*」と「*/」で囲まれた領域が、コメントになります。

```
/*  コメント  */

/*
  このように
  複数行にわたって
  書くこともできます。
*/
```

また、以下のように「//」を記述することで、その行の「//」より右側をコメントとすることができます。この 1 行コメントはもともと C++に搭載されていたものですが、1999 年に定められた規格「C99」において、C 言語にも正式に導入されました。

```
// 1 行コメント

// 囲う必要がないので楽ちん
```

コメントは、変数や関数の説明として利用されるだけでなく、デバッグ（バグを取り除く作業）において、一時的に一部のソースコードを無効化する際にも活躍します。コメントをうまく活用し、コード記述・開発の効率化を目指しましょう。

ソースコードには、適切なインデント（字下げ）を入れ、見た目を整えることも重要です。C 言語では、タブや半角スペース、改行が余分に入っていても、実行結果には影響しません。だからと言って、以下のように書いてしまってはとても読みにくいです。

```
 #include     <stdio.h>
int    main
      (  void)    {printf
( "Hello,world!\n"
    ) ;     return
       0   ;    }
```

※実行結果は、「1.6　文字の出力」のソースコードと同じになります。

これは極端な例ですが、多くのプログラミング初学者は、インデントによる整形の基本的なルールを守ることができていません。整形の基本的なルールとは、「波括弧で囲まれた領域は、均一のインデント（字下げ）を加える」という単純なものです。波括弧で囲まれた領域はブロックとも呼ばれ、一連の処理の塊を意味します。この塊が、どこから始まってどこで終わるのか、それをひと目で明らかにするために、インデントは加えられます。インデントとしては、半角スペース 4 個分、もしくは TAB がよく利用されます。よく見られるインデントの無いソースコードは、以下のようなものです。

```
#include <stdio.h>

int main(void) {
printf("Hello, world!\n");
return 0;
}
```

一方、適切なインデントが入ったソースコードは以下のようになります。

```
#include <stdio.h>

int main(void) {
    printf("Hello, world!\n");
    return 0;
}
```

これくらいの短いコードでは大きなな違いは見られませんが、後者のほうが、「main 関数の範囲」がひと目でわかると思います。今後、さまざまな処理によって波括弧の数はどんどん増えていきます。このとき、インデントによる整形のルールを遵守したコードとそうでないコードの間には、理解のしやすさに非常に大きな差が生じます。

1.9　課題

課題 1

あなたの好きな歌、詩、文章、もしくはあなたの自己紹介文を、綺麗に表示して下さい。

「ソースコード」と「実行結果」をそれぞれ載せること。

コラム 1　main 関数の書き方

main 関数の書き方について、「習った書き方と違う」「持っている本と違う」など、疑問に思われた方がいるかもしれませんので、ここで補足説明をしておきます。

実は、main 関数の書き方は、以下のようにさまざま存在します。

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

さらに、スペースの入れ方の流儀や * の位置などを考えると、嫌というほどの組み合わせになります（コラム 2 では、更に異なった表記方法も紹介します)。カーニハン＆リッチーによって書かれた C 言語のバイブル「The C Programming Language」では、「main()」が使われていることから、これに倣っている書籍が多く見受けられます。しかし、C 言語の仕様書の最終ドラフト（n1570）を見ると、記載されているのは以下の二つのみです。

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

※　`*argv[]` は `**argv` とすることも可

コンパイラによっては「int」や「void」を省略すると自動で挿入されることもありますが、それらは結局コンパイラ依存であり、決して安全とは言えません（ほとんどの場合安全ではありますが)。一方で、上記の二つの書き方は、現存するどのようなコンパイラでも問題なく動く、完璧に安全な main 関数と言えます。以上のことから、本書では「int main(void)」という書き方を利用していきます。後者の「int main(int argc, char *argv[])」との違いについては、第 5 章「関数」にて説明します。