平成 28 年度

C 言語演習用テキスト

東京工科大学

加納　徹

目次

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

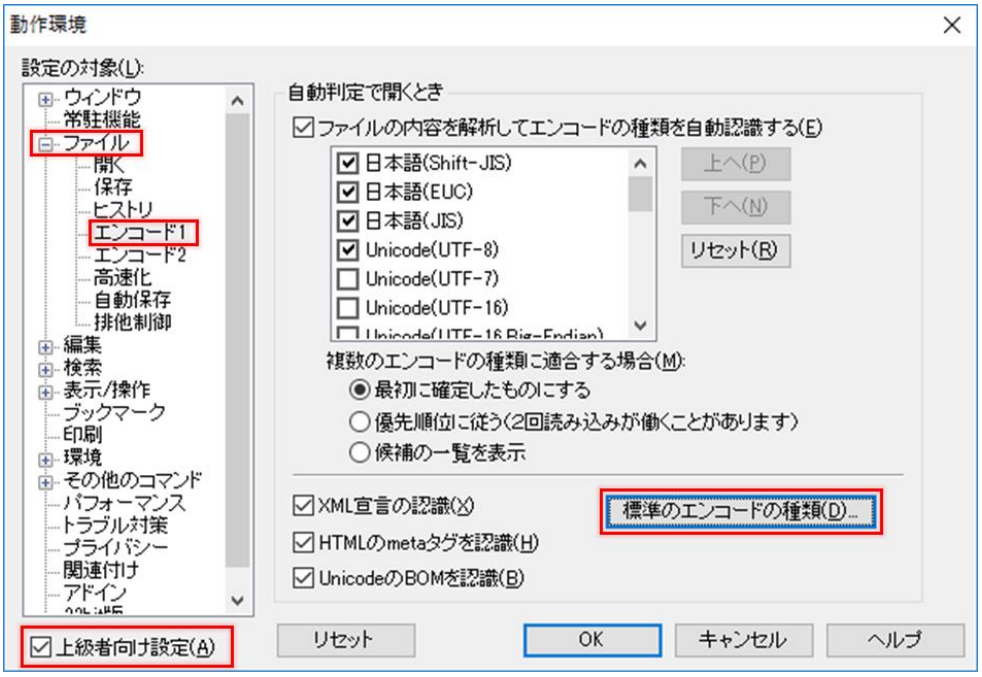
※2016 年 9 月 28 日時点での最新バージョンは、「hm864_signed.exe」です。

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。

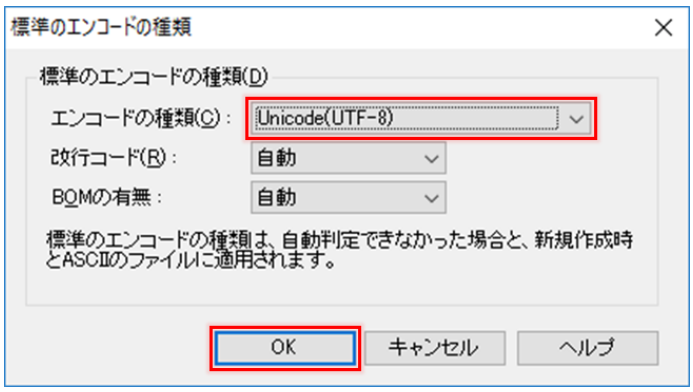
文字のエンコード設定

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。

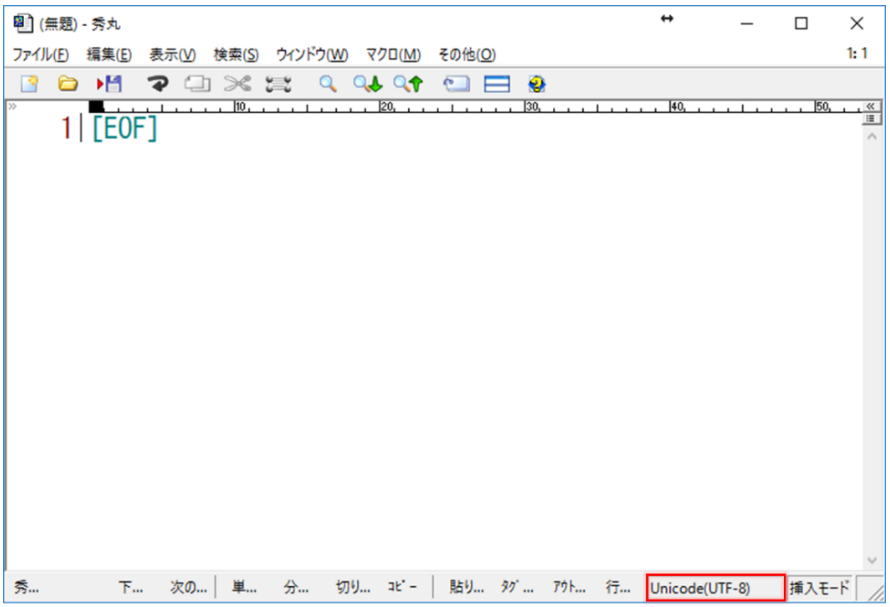
秀丸を立ち上げたら、メニューバーから［その他］→［動作環境］と進みます。動作環境ウィンドウが出たら、左下の［上級者向け設定］にチェックを入れ、［ファイル］→［エンコード 1］→［標準のエンコードの種類］と進みます。



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。

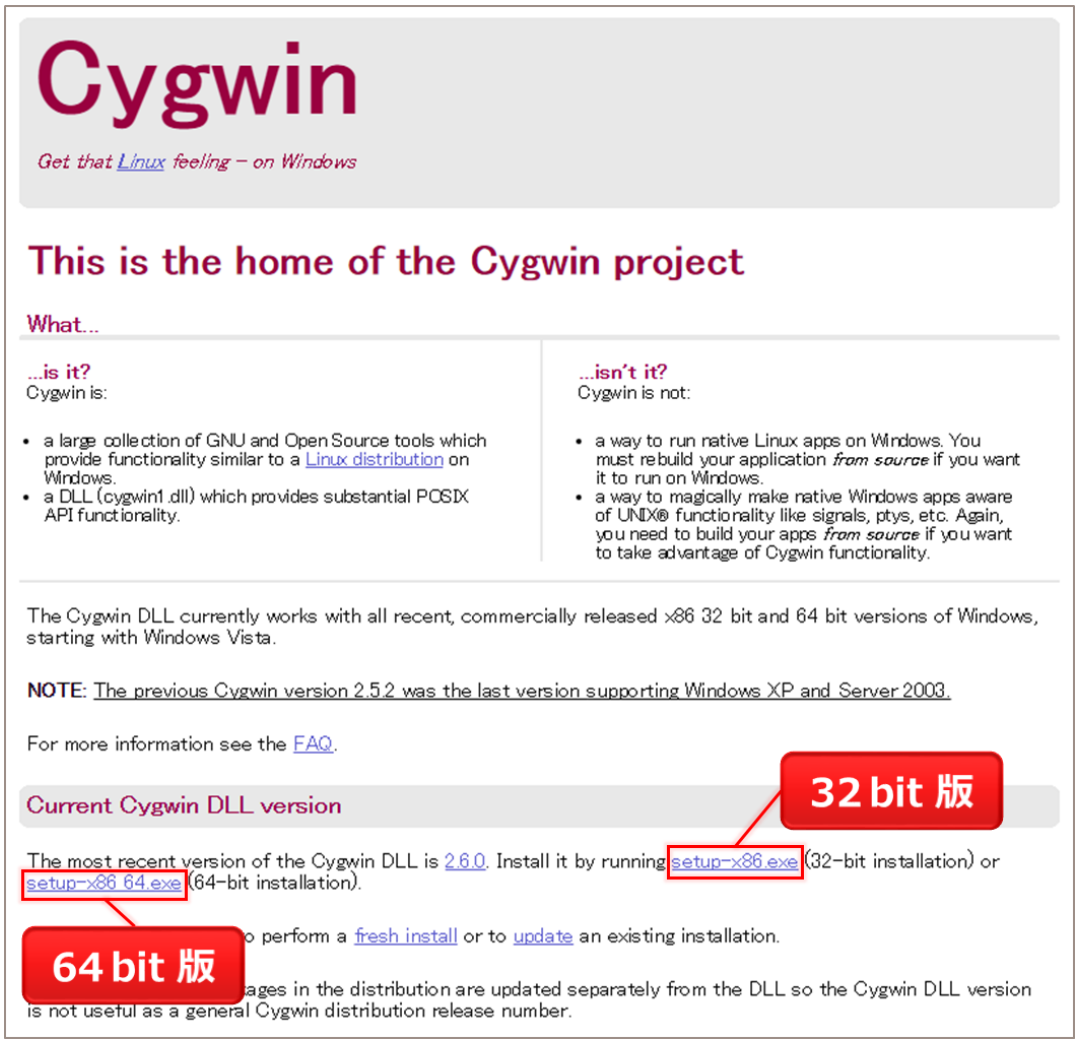
秀まるおのホームページから、［サポート］→［秀丸エディタフリー制度］→［アカデミックフリー個人］と進み、学校のメールアドレス（ac.jp で終わるもの）を用いて登録申請を行って下さい。もし秀丸のことが気に入って、大学卒業後も使っていきたいとなった場合は、お金を払いましょう。

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）。

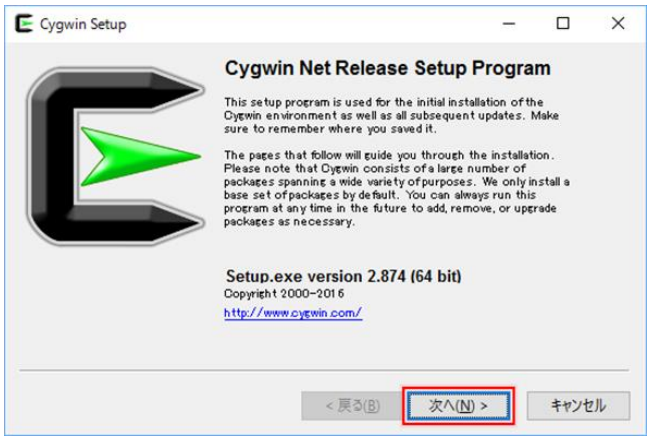
（Cygwin 公式ページ）

https://www.cygwin.com/

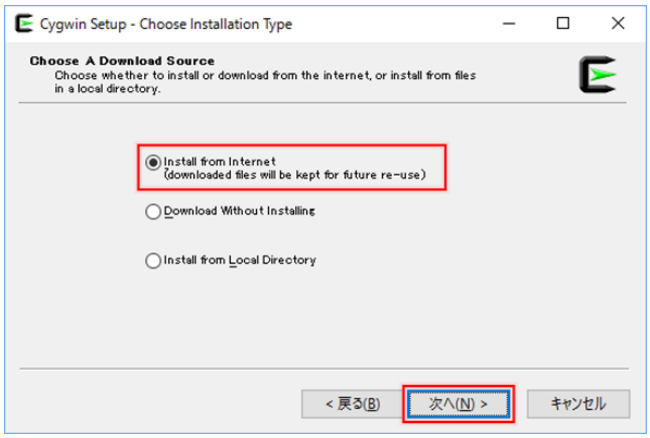


※2016 年 9 月 28 日時点での最新バージョンは、「2.6.0」です。

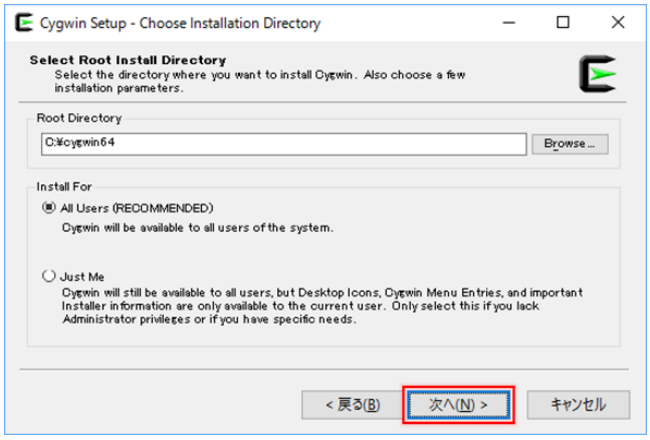
① インストーラを立ち上げると、次のような画面になります。[次へ] ボタンを押して進みましょう。



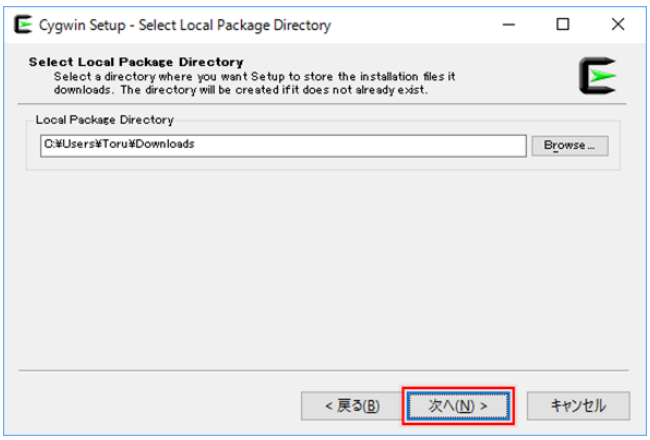
② データのダウンロード方法を選択します。[Install from Internet] がチェックされていることを確認し、[次へ] を押します。



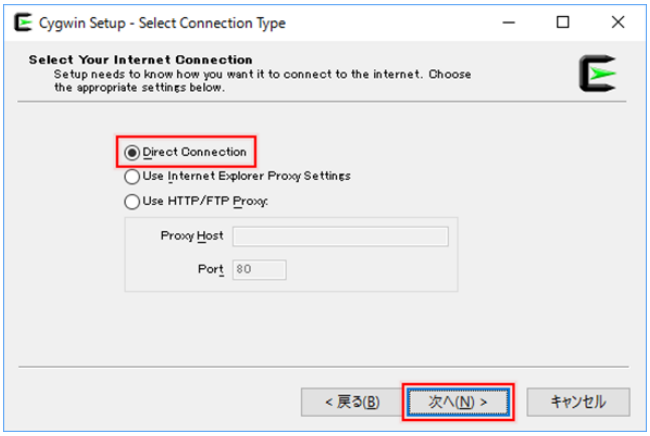
③ Cygwin のインストールフォルダ（Root Directory）を選択します。特にこだわりが無ければ、そのまま [次へ] を押します。



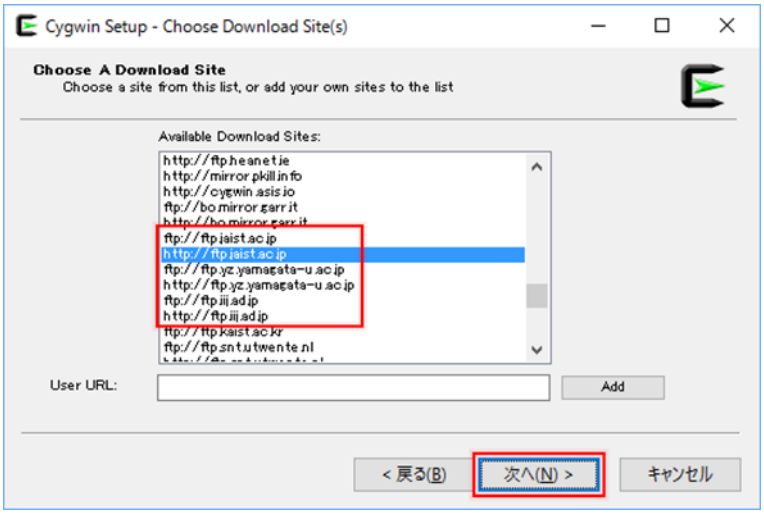
④ 一時的にインストールファイルを保存するフォルダを選択します。特にこだわりが無ければ、そのまま［次へ］を押します。ここで指定したフォルダに生成されたファイルは、Cygwin のインストール後に削除しても大丈夫です。



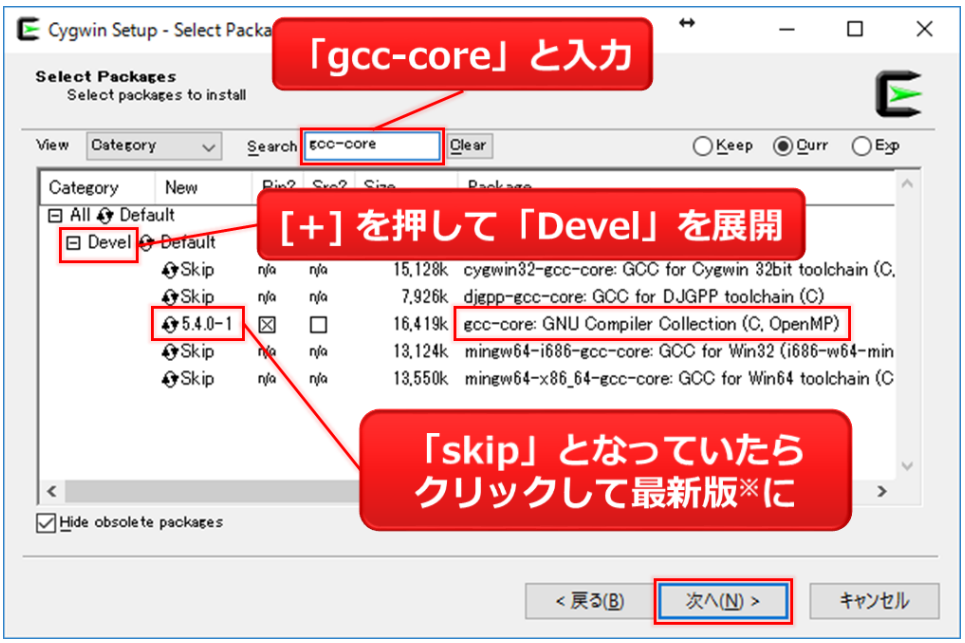
⑤ インターネット接続の方式を選択します。特にこだわりが無ければ、そのまま［次へ］を押します。



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。

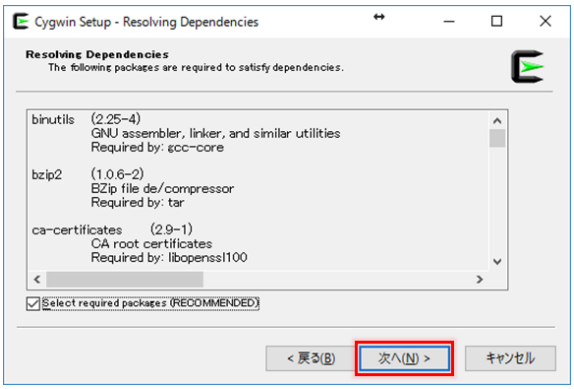


⑦ インストールするパッケージを選択します。「gcc」の核の部分だけインストールすれば良いので、[Search] と書かれた領域に「gcc-core」と入力し、フィルタリングします。[Devel] を展開して「gcc-core: GNU Compiler Collention (C, OpenMP)」を探し、[Skip] となっていたらクリックして最新版をインストールするように変更します。

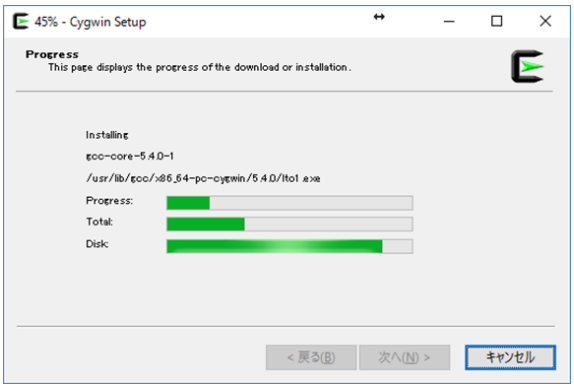


※2016 年 09 月 28 日時点での最新バージョンは 5.4.0-1 です。

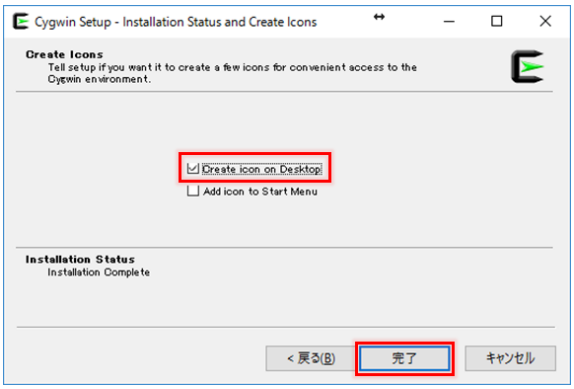
⑧ 依存性の問題を解消するため、追加で必要なパッケージを自動的に選択してくれます。そのまま［次へ］を押しましょう。



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。

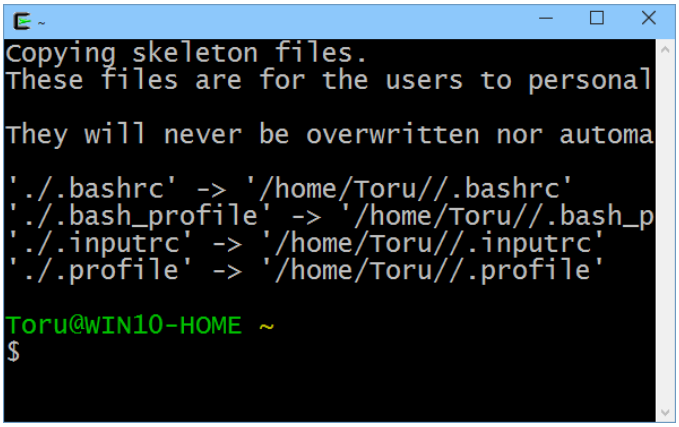
1.5　UNIX コマンドによる操作

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。

第 2 章　変数と演算

本章では、C 言語における変数と、その周辺の知識を説明します。

2.1　変数とは

変数とは、数字や文字などのデータを格納しておく、データの入れ物のようなものです。さまざまな種類の変数を扱うために、さまざまな変数の型（type）が用意されています。代表的な変数の型は、以下のとおりです。

型名	データの種類	扱える範囲
int	整数値	−2147483648 ～ 2147483647
long	倍長整数値	−9223372036854775808 ～ 9223372036854775807
float	実数値	有効数字 7 桁 $\pm 10^{-38}$ ～ 10^{38}
double	倍精度実数値	有効数字 15 桁 $\pm 10^{-308}$ ～ 10^{308}
char	文字	ASCII 文字（−128 ～ 127）

他にもさまざまな型がありますが、とりあえずは

「整数を扱いたいなら int」

「実数を扱いたいなら double」

「文字を扱いたいなら char」

の三つだけを覚えておけば、多くの場合問題ありません。

2.2　変数の宣言・代入・参照

変数をプログラムで使う場合、大きく「宣言」「代入」「参照」の三つの手順が必要になります。

変数の宣言

変数を使う場合、まずは名前をつける必要があります。これを「変数の宣言」と言います。変数の宣言は、以下のように、型名に続けて名前を記入します。

```
int number;   // number という名前の整数型変数の宣言
double value; // value という名前の実数型変数の宣言
char letter;  // letter という名前の文字型変数の宣言
int x, y;     // カンマで区切って同時に複数宣言することも可能
```

値の代入

変数に数字や文字などのデータを入れるには、代入演算子「=」を用います。ここで、「=」は数学の「等しい」という意味ではなく、「(右辺を左辺に) 代入」を意味することに注意して下さい。また、宣言された変数に初めて値を代入することを「**初期化**」と言います。初期化されていない変数を演算させたり、参照したりしようとすると、当然エラーが発生します。

```
1  // 変数の宣言
2  int x, y;
3  double value;
4  char letter;
5
6  // 値の代入 ( 初期化 )
7  x = 10;
8  y = -5;
9  value = 3.1415;
10 letter = 'P';
11
12 // 値の代入 ( 更新 )
13 x = y;
```

宣言した変数に対して、7〜10 行目でそれぞれ値を代入しています。文字型の変数 letter には、1 文字だけを格納することができます。1 文字（文字定数）は、上記ソースコードのようにシングルクォーテーション ‘’ で囲って表現します。また、13 行目では変数 x の値を y の値で上書きしています。「=」を等号と考えた場合、「x = y」は間違った等式になりますが、C 言語では「x に y の値を代入する」という意味になり、間違いではありません。この代入の前、「x = 10」「y = -5」でしたが、代入後は「x = -5」「y = -5」となります。

上のソースコードでは、「変数の宣言」と「値の代入 (初期化)」を独立して行っています。これらの操作は、以下のように同時に行うこともできます。覚えておきましょう。

```
1  // 変数の宣言と初期化
2  int x = 10, y = -5;
3  double value = 3.1415;
4  char letter = 'P';
```

値の参照

変数に入っているデータを出力するためには、第 1 章でも使った「printf」関数を用います。「printf」で変数を出力するには、以下の形で記述します。

```
printf(“フォーマット指定子”, 変数);
```

フォーマット指定子は「%」からはじまる書式で、実行時に変数に置き換えられます。フォーマット指定子としては、以下のようなものがあります。

型名	データの種類	指定子
int	整数値	%d
long	倍長整数値	%ld
float	実数値	%f
double	倍精度実数値	%lf
char	文字	%c

実際に値を参照するコードを書いて確認しましょう。

演習 2-1 変数の表示の練習（ex2-1.c）

以下のソースコードを参考に、年齢・身長・血液型を表示するプログラムを作成して下さい。

```c
#include <stdio.h>

int main(void) {
    // 変数の宣言と初期化
    int age = 29;
    double height = 175.8;
    char blood = 'A';

    // 変数の参照
    printf("   age = %d\n", age);
    printf("height = %lf\n", height);
    printf(" blood = %c\n", blood);

    return 0;
}
```

演習 2-1 実行例

```
$ ./a
    age = 29
height = 175.800000
 blood = A
```

変数表示の詳細設定

「printf」で変数を表示させる際、表示する桁数などを指定することができます。桁数の指定は、「%[全体の幅]d」や「%[全体の幅].[小数点以下の幅]」といった形で行います。また、「%」の直後に「0」を書くと「0 詰め」、「-」を書くと「左寄せ」、「+」を書くと「符号表示」になります。実際に以下のコード例を見てみましょう。

例 2-1

```c
#include <stdio.h>

int main(void) {
    // 変数の宣言と初期化
    int age = 29;
    double height = 175.8;
    char blood = 'A';

    // 変数の参照（幅 7 桁、小数以下 2 桁）
    printf("[%7d]\n", age);
    printf("[%7.2lf]\n", height);
    printf("[%7c]\n\n", blood);

    // 変数の参照（幅 7 桁、小数以下 2 桁、0 詰め）
    printf("[%07d]\n", age);
    printf("[%07.2lf]\n", height);
    printf("[%07c]\n\n", blood);

    // 変数の参照（幅 7 桁、小数以下 2 桁、左寄せ）
    printf("[%-7d]\n", age);
```

```
21      printf("[%-7.2lf]\n", height);
22      printf("[%-7c]\n\n", blood);
23
24      // 変数の参照（幅 7 桁、小数以下 2 桁、符号の表示）
25      printf("[%+7d]\n", age);
26      printf("[%+7.2lf]\n", height);
27      printf("[%7c]\n\n", blood);
28
29      return 0;
30  }
```

例 2-1 実行結果

```
$ ./a
[     29]
[ 175.80]
[      A]

[0000029]
[0175.80]
[000000A]

[29     ]
[175.80 ]
[A      ]

[    +29]
[+175.80]
[      A]
```

このように、変数表示の詳細を指定すれば、桁数が異なる変数も綺麗に揃えて出力することができます。

演習 2-2 変数を揃えて表示（ex2-2.c）

例 2-1 のソースコードを参考に、演習 2-1 のソースコードの結果を右揃えで出力して下さい。

演習 2-2 実行例

```
$ ./a
   age =    29
height = 175.8
 blood =     A
```

2.3　変数の命名規則

変数の名前の付け方には、「守らないといけないルール」と「守ったほうが良いルール」があります。それぞれを見ていきましょう。

守らないといけないルール

以下のルールは必ず守る必要のあるもので、守れていない場合はエラーが発生します。

- 半角のアルファベット、数字、アンダースコア（_）のみを用いる。
- 先頭は必ずアルファベットかアンダースコアを用いる（数字は不可）。
- 予約語は使用できない。

ここで予約語とは、C 言語ですでに意味を持っているワードのことで、「int」「double」「if」「switch」「for」「do」「while」「goto」「return」などがあります。

守ったほうが良いルール

以下に示すルールは、C 言語で変数名を付ける際のマナーのようなものです。必ず守る必要があるわけではありませんが、守ることで読みやすさや理解のしやすさが向上します。

- a, b, c など、意味のわかりにくい名前はつけない。
- アルファベットは全て小文字を用いる（定数の場合は全て大文字）。
- 複合語は単語の間にアンダースコアを入れる（スネークケース）。

名前の付け方には、「スネークケース」「キャメルケース」「パスカルケース」と呼ばれるものがあります。スネークケースは「snake_case」のように、全て小文字で単語の間にアンダースコアを入れる表記法、キャメルケースは「camelCase」のように、小文字で始まり単語の切れ目で大文字を用いる表記法、パスカルケースは「PascalCase」のように、単語の始ま

りを全て大文字にする表記法です。一般的に、C 言語の変数名にはスネークケースによる記法が用いられます。以上のことから、「利用不可な例」「悪い例」「良い例」をまとめて表にすると、次のようになります。

利用不可な例	悪い例	良い例
c-3	a, b, c	pos_x, pos_y, pos_z
!?#	a1_1, a1_2	position, velocity
(^-^)	l1, ll, l1I	num, value, count
2016_Oct	getdata	get_data
return	GETTEXTDATA	get_text_data

2.4　定数

「変数」はその名のとおり、プログラム中で値が変化する可能性があるデータです。一方、プログラム中に値が決して変化しない「定数」というデータを利用することもできます。定数は、通常「include 文」の下（main 関数の外）に、「#define 定数名 値」といった形で定義します。行の終わりを表すセミコロン「;」は付けません。また、定数は変数との区別と容易にするため、一般的に全て大文字で書きます。以下のコードを見て使用例を確認しましょう。

例 2-2

```c
#include <stdio.h>
#define RATE 101.338 // 為替レート

int main(void) {
    printf("現在のレートは 1 ドル = %lf 円です。\n", RATE);
    return 0;
}
```

例 2-2 実行結果

```
$ ./a
現在のレートは 1 ドル = 101.338000 円です
```

以上のように、定数の参照は変数と同様に行うことができます。ただし、変数とは違い、プログラムの途中で値を変更することはできません。

2.5　演算

C 言語での基本的な演算について、見ていきましょう。算術演算を行うための代表的な「算術演算子」を、以下に示します。

演算子	役割	例	意味
+	加算	z = x + y;	x に y を足した値を z に代入
-	減算	z = x - y;	x から y を引いた値を z に代入
*	乗算	z = x * y;	x に y を掛けた値を z に代入
/	除算	z = x / y;	x を y で割った値を z に代入
%	剰余算	z = x % y;	x を y で割った余りを z に代入

演習 2-3 算術演算子による整数の計算（ex2-3.c）

int 型の変数 x, y, z を用意し、x と y に適当な値を入れ、上の表に示した 5 つの算術演算の結果を表示して下さい。

演習 2-3 コード例

```c
#include <stdio.h>

int main(void) {
    int x = 18, y = 4;

    printf("x = %d, y = %d\n", x, y);
    printf("x + y = %d\n", x + y);
    printf("x - y = %d\n", x - y);
    printf("x * y = %d\n", x * y);
    printf("x / y = %d\n", x / y);
    printf("x %% y = %d\n", x % y);

    return 0;
}
```

※「%」はフォーマット指定子で使う記号であるため、
「%」のみを出力したい場合は「%%」と書きます。

演習 2-3 実行例

```
$ ./a
x = 18, y = 4
x + y = 22
x - y = 14
x * y = 72
x / y = 4
x % y = 2
```

ここで、「18 ÷ 4」の値が「4」になっていることに注意して下さい。このように、int 型（整数型）同士の演算結果は、小数点以下が切り捨てられ、必ず整数値になります。また、double 型（実数型）での計算を行ったとしても、int 型の変数に格納しようとした場合、同様に整数化されます。

演習 2-4 算術演算子による実数の計算（ex2-4.c）

double 型の変数 x, y, z を用意し、x と y に適当な値を入れ、「加算」「減算」「乗算」「除算」の結果を表示して下さい（実数値の「剰余算」は「%」演算子で行うことができません）。

演習 2-4 実行例

```
$ ./a
x = 18.00, y = 4.00
x + y = 22.00
x - y = 14.00
x * y = 72.00
x / y = 4.50
```

算術演算子 ＋ 代入演算子

算術演算子は、以下のように代入演算子と組み合わせることができます。

演算子	役割	例	意味
+=	加算代入	x += y;	x に y を足した値を x に代入
-=	減算代入	x -= y;	x から y を引いた値を x に代入
*=	乗算代入	x *= y;	x に y を掛けた値を x に代入
/=	除算代入	x /= y;	x を y で割った値を x に代入
%=	剰余算代入	x %= y;	x を y で割った余りを x に代入

これらの演算は「x＝x＋y」のように書いても同じ結果となります。しかし、加算代入演算子などを用いることで、コードを短く整理することができます。

インクリメント/デクリメント演算子

プログラムでは、変数を 1 ずつ増やしたり減らしたりすることがよくあります。そのようなときは、インクリメント演算子（++)、デクリメント演算子（--）を用いることができます。実際には、以下のように整数型の変数に繋げて用います。

```
1 int index = 0;
2
3 index++; // index に 1 を足す
4 index--; // index から 1 を引く
```

これら演算子は「++index」「--index」のように、変数の前に置くこともできます。この例のように単独で用いる場合は全く同じ結果になりますが、代入演算子「=」や比較演算子（第 3 章参照）と同時に用いると、異なる挙動を示すので注意が必要です。

```
1 int n, index = 0;
2
3 n = index++; // n に index を代入してから 1 を足す
4 n = ++index; // index に 1 を足してから n に代入する
```

インクリメント/デクリメント演算子は、通常の加算/減算処理よりも高速に動作します。そのため、何度も繰り返し 1 を加算したり減算したりする場合は、これら演算子を用いるようにしましょう。

2.7　キーボードからの入力

インタラクティブなプログラムを作成するには、キーボードなどからの入力を受け取る必要があります。データの入力には、「scanf」関数を用います。実際に、int 型の変数と double 型の変数を受け取る例を見てみましょう。

例 2-3

```c
#include <stdio.h>

int main(void) {
    int age;
    double height;

    printf("年齢を入力して下さい: ");
    scanf("%d", &age);

    printf("身長を入力して下さい: ");
    scanf("%lf", &height);

    printf("age = %d, height = %.2lf\n", age, height);

    return 0;
}
```

例 2-3 実行結果

```
$ ./a
年齢を入力して下さい: 29
身長を入力して下さい: 175.8
age = 29, height = 175.80
```

このように「scanf」では、読み取りたい変数の型のフォーマット指定子（「printf」のものと同じ）を記入します。そしてカンマで区切り、値を格納する変数に「&」記号を付けて記入します。「&」は変数のアドレス（住所）を表す記号で、値を格納する場所をコンピュータに教えてあげています（詳細は第 6 章「配列とポインタ」にて）。また、「scanf(“%d %lf”, &age, &height);」と書いて、複数の変数を同時に読み取ることも可能です。ただし、入力時には数値をスペースで区切る必要があります。

演習 2-5 データ入力の練習（ex2-5.c）

「scanf」を用いて生まれ年と現在の年を西暦で受け取り、年齢を出力するプログラムを作成して下さい。

演習 2-5 実行例

```
$ ./a
生まれ年（西暦）を入力して下さい：1987
現在の年（西暦）を入力して下さい：2016
2016 年、あなたは今年 29 歳です。
```

2.6　課題

課題 2-1

int 型、double 型、char 型の変数に対し、「整数値」「実数値」「文字」をそれぞれ代入するとどのようになるか、9 通りの結果をまとめて下さい。また、char 型の変数をフォーマット指定子「%d」で表示をし、その結果について考察をして下さい。

課題 2-2

7! と17! の値を計算して表示して下さい。また、結果について考察をして下さい。

（「!」は階乗を意味します。例えば、$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$ となります。）

課題 2-3

円錐の底面の半径と高さを入力すると、体積を計算して表示するプログラムを作成して下さい。円周率は定数として定義しましょう。また、真値との誤差について、考察・検討をして下さい。

第 3 章　条件分岐

本章では、プログラムの流れに分岐を作る「if」文と「switch」文を説明します。

3.1　if 文

「if」文は英語での意味のとおり、「もしも・・・ならば」という条件分岐を表します。条件分岐の例として、次の文を考えてみましょう。

もしも成績が 60 点以上ならば、単位がもらえる

皆さんに今後待ち受ける未来として、「成績が 60 点以上で単位がもらえる」と「成績が 59 点以下で単位がもらえない」の二通りが存在します。このように、特定の条件によって分岐が生じる（異なった未来が待ち受けている）ことを、**条件分岐**と言います。「降水確率が 30%以上なら傘を持って出かける」「身長が 130 cm 以上ならジェットコースターに乗れる」など、世の中のあらゆることは条件分岐の組み合わせでできています。

実際に、以下の仕様を持つプログラムを見てみましょう。

- 成績の入力を受け付ける
- 成績が 60 点以上なら「単位がもらえます。」と表示する

例 3-1

```c
#include <stdio.h>

int main(void) {
    int score;

    printf("成績を入力して下さい: ");
    scanf("%d", &score);

    if (score >= 60) {
        printf("単位がもらえます。\n");
    }

    return 0;
}
```

例 3-1 実行例

```
$ ./a
成績を入力して下さい: 80
単位がもらえます。

$ ./a
成績を入力して下さい: 50
```

このように if 文は、「条件式」と、条件式を満たしていた際の「処理」で構成されます。「条件式」には、「score が 60 点以上」を表す「score >= 60」が入っています。また、「処理」の範囲は main 関数と同じように波括弧 { } で囲って表します。ただし、処理が 1 行である場合には、波括弧を省略することも可能です。

```
if ( 条件式 ) {
    // 条件式を満たしていた場合の処理
}

if ( 条件式 )
    // 処理が 1 行であれば波括弧は省略可
```

3.2　条件の表し方

前節の例では、「以上」を表す記号として「>=」を用いました。このように、二つの数値の関係を表す記号を「関係演算子」（もしくは「比較演算子」）と言います。

関係演算子

関係演算子は、if 文や for, while 文（第 4 章）の条件式の中でとてもよく使われます。他にどのような関係演算子があるのか、以下の表を見てみましょう。

演算子	意味	数学記号での表記
`n > 60`	n が 60 よりも大きい	$n > 60$
`n < 60`	n が 60 よりも小さい	$n < 60$
`n >= 60`	n が 60 以上	$n \geq 60$
`n <= 60`	n が 60 以下	$n \leq 60$
`n == 60`	n が 60 と等しい	$n = 60$
`n != 60`	n が 60 と等しくない	$n \neq 60$

関係演算子を用いることで、数学で行うものと同じように、変数や数値の比較をすることが可能です。ただし、C 言語のプログラミングで使用可能な記号の数はかなり制限されているため、「≥」は「>=」、「≤」は「<=」、「≠」は「!=」のように表します。また、C 言語における「等しい」の関係演算子は、「==」と表すことに注意して下さい。第 2 章でも述べたように、C 言語上で「=」は代入演算子であり、「右辺のものを左辺に代入する」という意味になります。

論理演算子

if 文の条件式の構成要素には、「関係演算子」の他に「論理演算子」というものがあります。例えば、「成績が 60 点以上かつ 70 点未満の場合」のように、二つ以上の条件をまとめて扱いたい場合があります。数学では「$60 \leq n < 70$」のように書くことができますが、C 言語では「`60 <= n < 70`」のようにまとめて書くことはできません。このようなときは、以下の表に示す論理演算子を使います。

演算	意味	使用例	使用例の意味
`&&`	どちらも真	`60 <= n && n < 70`	n が 60 以上かつ 70 未満
`||`	どちらかが真	`n < 60 || 70 <= n`	n が 60 未満または 70 以上
`!`	否定	`!(n < 60)`	n が 60 以上でない（60 未満）

すなわち、「成績が 60 点以上かつ 70 点未満の場合」という条件分岐を作りたい場合は、「成績が 60 点以上」と「成績が 70 点未満」の二つの条件に分け、「&&」を用いて以下のようにします。

```
if (60 <= n && n < 70)
```

関係演算子と論理演算子を組み合わせれば、さまざまな条件を作ることが可能になります。また、数学で「&&」は「∩（論理積）」、「||」は「∪（論理和）」のことであり、それぞれ「アンド（and）」「オア（or）」と呼ばれます。

3.3　if〜else 文

ここまでで、if 文を使って「もし〜ならば、この処理をする」というプログラムが書けるようになりまりた。しかし、「そうでなければ、この処理をする」というプログラムを書くことも頻繁にあります。そのような場合は、「else」文を使います。実際の使用例を見てみましょう。

例 3-2

```c
#include <stdio.h>

int main(void) {
    int score;

    printf("成績を入力して下さい: ");
    scanf("%d", &score);

    if (score >= 60) {
        printf("単位がもらえます。\n");
    } else {
        printf("不可です。\n");
    }

    return 0;
}
```

例 3-2 実行例

```
$ ./a
成績を入力して下さい: 80
単位がもらえます。

$ ./a
成績を入力して下さい: 50
不可です。
```

例 3-1 からの変更点は、12〜14 行目の部分のみです。このプログラムは、成績が 60 点以上であるかどうかを確認し、そうであれば「単位がもらえます。」を、そうでない場合は「不可です。」を表示するものとなっています。このように、if 文の後に else 文を入れることで、if 文の条件を満たさなかった場合の処理を書くことができます。

```
if (条件式) {
    // 条件式を満たす場合の処理
} else {
    // 条件式を満たさない場合の処理
}
```

また、「else if」文を入れることで、以下のように細かい条件分岐を作ることもできます。else if 文は幾つでも追加することが可能です。

```
if (条件式 A) {
    // 条件式 A を満たす場合の処理
} else if (条件式 B){
    // 条件式 B を満たす場合の処理
} else if (条件式 C){
    // 条件式 C を満たす場合の処理
} else {
    // いずれの条件式も満たさない場合の処理
}
```

if〜else if〜else 文を使って、以下の仕様を持つプログラムを作るとどうなるか、見てみましょう。

- 成績の入力を受け付ける
- 成績が 70 点以上なら「余裕で単位がもらえます。」と表示する
- 成績が 60 点以上 70 点未満なら「ぎりぎり単位がもらえます。」と表示する
- 成績が 60 点未満なら「不可です。」と表示する

例 3-3

```
1  #include <stdio.h>
2
3  int main(void) {
4      int score;
5
6      printf("成績を入力して下さい: ");
7      scanf("%d", &score);
8
9      if (score >= 70) {
10         printf("余裕で単位がもらえます。\n");
11     } else if (score >= 60) {
12         printf("ぎりぎり単位がもらえます。\n");
13     } else {
14         printf("不可です。\n");
15     }
16
17     return 0;
18 }
```

ここで注目してもらいたいのは、12 行目の条件文です。この else if 文では単に「score が 60 点以上」という条件を書いていますが、この分岐点に来るということは直前の if 文で「score が 70 点以上」という条件を満たしていないので、必然的に「score は 70 未満」ということになります。すなわち、これだけで「成績が 60 点以上 70 点未満」という条件を作ることができるというわけです。もちろん、論理演算子「&&」を用いて明示的に「`60 <= score && score < 70`」を書いても構いません。

ホワイトボックステスト（網羅率の検証）

プログラムを書いた場合、必ず正しく動作するかをチェックしなければなりません。特に if 文や後述の switch 文を使う場合はプログラムに分岐ができるので、処理の分岐が正しく行われているか、各処理が正しく動作するかを検証する必要があります。この検証はホワイトボックステストと呼ばれ、テストの有効性は網羅率で表されます。代表的な「網羅性」を以下に示します。

- 分岐網羅・・・全ての分岐を必ず一度は実行すること
- 条件網羅・・・全ての条件の組み合わせを必ず一度は実行すること

以下のソースコードを考えてみましょう。

```
if (a < 5 && b < 10) {
    処理 A;
} else {
    処理 B;
}
```

このとき、「a = 0, b = 6」と「a = 6, b = 11」などの 2 つのケースでプログラムの動作を確認すれば、処理 A と処理 B が実行されることになり、分岐網羅を満たします。しかし、2 つの条件式の真偽の組み合わせは、4 通りあります。条件網羅を満たすには、さらに「a = 0, b = 11」「a = 6, b = 6」などのケースを試し、全ての条件式の真偽の組み合わせで動作を確認する必要があります。プログラムは複雑になるにつれ、エラーやバグが生じやすくなります。そのような自体を防ぐため、ソースコードが書けたら少なくとも「分岐網羅」、できれば「条件網羅」を満たすよう、ホワイトボックステストを実施するようにしましょう。

演習 3-1 (ex3-1.c)

例 3-3 を参考に、以下の仕様を持つプログラムを作成しなさい。

- 成績の入力を受け付ける
- 成績が 0 点未満または 100 点より大きい場合、
  「正しい値を入力して下さい。」と表示する
- 成績が 90 点以上 100 点以下なら「S 単位がもらえます。」と表示する
- 成績が 80 点以上 90 点未満なら「A 単位がもらえます。」と表示する
- 成績が 70 点以上 80 点未満なら「B 単位がもらえます。」と表示する
- 成績が 60 点以上 70 点未満なら「C 単位がもらえます。」と表示する
- 成績が 60 点未満なら「不可です。」と表示する

3.4　switch～case 文

プログラムの分岐を作るには、前節までで紹介した「if」文の他に、「switch」文があります。switch 文は、分岐の数が多いときによく使われる構文で、書式は以下に示すとおりです。

```
switch ( 式 ) {
    case 定数式 A:
        // 式と定数式 A が一致した場合の処理
        break;
    case 定数式 B:
        // 式と定数式 B が一致した場合の処理
        break;
    default:
        // 式がどの定数式とも一致しなかった場合の処理
        break;
}
```

書式を見ただけではわかりにくいと思うので、実際に if 文で書かれたソースコードを、switch 文に書き直すことを考えてみましょう。以下に、if 文による分岐を実装したコードを示します。

例 3-4

```
1  #include <stdio.h>
2
3  int main(void) {
4      char c;
5
6      printf("あなたの好きなスポーツは？\n");
7      printf("  (a) 野球\n");
8      printf("  (b) サッカー\n");
9      printf("  (c) 特にない\n");
10     scanf("%c", &c);
11
12     if (c == 'a') {
13         printf("やっぱり日本人なら野球ですよね。\n");
```

```
    } else if (c == 'b') {
        printf("サッカー、格好いいですよね。\n");
    } else if (c == 'c') {
        printf("特にないのですね。\n");
    } else {
        printf("不正な文字が入力されました。\n");
    }

    return 0;
}
```

このプログラムは、好きなスポーツに対応するアルファベットの入力を受け取り、結果を表示するものです。これを switch 文に書き換えると、次のようになります。

例 3-5

```
#include <stdio.h>

int main(void) {
    char c;

    printf("あなたの好きなスポーツは？\n");
    printf("  (a) 野球\n");
    printf("  (b) サッカー\n");
    printf("  (c) 特にない\n");
    scanf("%c", &c);

    switch (c) {
        case 'a':
            printf("やっぱり日本人なら野球ですよね。\n");
            break;
        case 'b':
            printf("サッカー、格好いいですよね。\n");
            break;
        case 'c':
            printf("特にないのですね。\n");
```

```
21              break;
22          default:
23              printf("不正な文字が入力されました。\n");
24              break;
25      }
26
27      return 0;
28  }
```

if 文のコードと比べてソースコードの行数は増えてしまっていますが、シンプルで読みやすい構造になっていることがわかると思います。重要な行を以下で説明していきます。

11 行目：　switch 文ではこのように、switch の後に括弧（）で何らかの式を囲います。式には整数を結果とするもののみが利用可能です（char 型は整数と同様に扱われます）。

12 行目：　分岐は case 文で作っていきます。case の後に半角スペースを空けて定数式を記述すると、switch の式と case の定数式が一致したとき、case の中に書かれたコードが実行されます。また、case 文の行はセミコロン「;」ではなくコロン「:」で終わることに注意して下さい。

14 行目：　case 文の中の処理が終わったら、break 文を用いて switch の { } から抜け出します。一度 case 文の中に入った場合、break 文で抜け出さない限り、それ以降に書かれているすべての case 文（および default 文）の処理が実行されます。

21 行目：　どの case にも当てはまらない場合の処理は、default 文の中に記述します。case 文での default 文は、if 文で言うところの else 文に相当します。

演習 3-2 (ex3-2.c)

例 3-5 のソースコードを元に、「野球」「サッカー」以外のスポーツ（例えばテニス）も選択できるできよう書き換えて下さい。また、break 文が無い場合にどのような挙動を示すか、実際に確認してみて下さい。

演習 3-3 (ex3-3.c)

演習 3-2 のソースコードを、大文字のアルファベットの入力にも対応したプログラムに改良して下さい。

ヒント： 以下のようなコードを書くと、例 3-5 の 14 行目のコード説明にある性質により、'a'を入力しても'A'を入力しても同じ結果を表示することができます。

```
case 'A':
case 'a':
    printf("やっぱり日本人なら野球ですよね。\n");
    break;
```

（補足： if 文を使って書く場合は、「if (c == 'A' || c == 'a')」のようにします。）

3.5　課題

課題 3-1

整数値の入力を受け付け、奇数であれば「奇数です。」、偶数であれば「偶数です。」と表示するプログラムを作って下さい。(if 文を利用)

ヒント： 偶数は 2 で割った余りが 0 になります。

課題 3-2

生まれ年を西暦で入力すると、その年の干支を表示するプログラムを作って下さい。(switch 文を利用)

ヒント： 西暦を 12 で割った余りが 0 のとき、干支は申です。

コラム 2　インデントスタイル

main 関数や if 文を利用する際、波括弧 {} で囲われた中身にインデント（字下げ）を入れ、処理の範囲をわかりやすくしてきました。このインデントを入れるスタイルは、実にさまざまなものが提案されています。

まだ

第 4 章　繰り返し処理

同じような処理を繰り返し実行したい場合は、「for」文もしくは「while」文を用います。それぞれについて、使い方を見ていきましょう。

4.1　for 文

for 文は、ある処理を指定した回数だけ繰り返す際に、よく用いられます。英語で「3 年間」などを表すときに「for three years」と言いますが、for 文の for はこれと同じ意味を持ちます。for 文の書式は以下に示すとおりです。

```
for ( 初期化 ; 条件式 ; 次の処理 ) {
    // 繰り返し実行する処理
}
```

このように、for 文では繰り返す回数を「初期化」「条件式」「次の処理」の 3 つの要素で表します（それぞれの要素はセミコロン“;”で区切ります)。文章ではプログラムの流れのイメージが湧きにくいと思いますので、実際のコード例を見てみましょう。以下のソースコードは、0 から 9 までの数字を printf 関数で表示するプログラムです。

例 4-1 数字の 0 から 9 を for 文で表示

```
1  #include <stdio.h>
2
3  int main(void) {
4      int i;
5      for (i = 0; i < 10; i++) {
6          printf("%d\n", i);
7      }
8      return 0;
9  }
```

4 行目： 繰り返し回数を数えるための整数型変数 i を宣言しています（i は integer もしくは index の頭文字を意味します)。

5 行目： for 文による繰り返し処理です。

「初期化」 変数 i に 0 を入れて初期化しています。

「条件式」 変数 i が 10 未満の間、波括弧の処理を繰り返します。

「次の処理」 波括弧の処理が終わったら、繰り返す前に実行する処理です。ここでは「i++」（インクリメント演算）を用いて、i に 1 を加算しています。

6 行目： 繰り返す処理の内容です。printf 関数で i の値を表示しています。

例 4-1 実行結果

```
$ ./a
0
1
2
3
4
5
6
7
8
9
```

ここでは 0 から 9 までの数字を表示しましたが、1 から 10 までの数字を表示したい場合は、以下のように表示する際に変数 i に 1 を加えてあげます。

数字の 1 から 10 を for 文で表示する例①

```
int i;
for (i = 0; i < 10; i++) {
    printf("%d\n", i + 1);
}
```

また、変数の値自体に 1 から 10 を入れたい場合、もしくは繰り返しの範囲を 1 から 10 と明示的に表したい場合は、for 文の「初期化」と「条件式」を以下のように修正します。

数字の 1 から 10 を for 文で表示する例②

```
int i;
for (i = 1; i <= 10; i++) {
    printf("%d\n", i);
}
```

もう一つ、簡単で実用的な for 文の例を見てみましょう。次のソースコードは、1 から 10 までの整数の和を求めるプログラムです。

例 4-2 正整数の 1 から 10 までの和を計算

```c
#include <stdio.h>

int main(void) {
    int i, sum = 0;
    for (i = 1; i <= 10; i++) {
        sum += i;
    }
    printf("1 から 10 までの和は %d です。\n", sum);
    return 0;
}
```

4 行目で、計算結果を格納するための変数 sum を用意し、6 行目で i の値を sum に加算していきます。for 文で i は 1 から 10 になるまで繰り返されるので、sum には 1 から 10 までの和が入ります。

例 4-2 実行結果

```
$ ./a
1 から 10 までの和は 55 です。
```

演習 4-1 (ex4-1.c)

正の整数値の入力を受け取り、その約数をすべて表示するプログラムを作って下さい。余力があれば、約数の個数と合計値も表示してみましょう。

ヒント：剰余（割った余り）が 0 であるかどうかで、約数かどうかを判定できます。

演習 4-1 実行例

```
$ ./a
正整数を入力して下さい: 27
27 の約数は、
1
3
9
27
の 4 個で、合計値は 40 です。
```

for 文の入れ子

for 文は入れ子の形でもよく使われます。入れ子はネスト（Nesting）とも言われ、プログラムの構造が再帰的に繰り返されることを表します。例えば、for 文の処理の中にまた for 文があれば、それは入れ子構造です。以下の for 文 n 入れ子構造を持つプログラムを見て、どのような結果になるか予想してみましょう。

例 4-3 入れ子による簡易グラフの表示

```c
#include <stdio.h>

int main(void) {
    int i, j;
    for (i = 1; i <= 10; i++) {
        printf("%2d: ", i);
        for (j = 0; j < i; j++) {
            printf("*");
        }
        printf("\n");
    }
    return 0;
}
```

例 4-3 実行結果

```
$ ./a
 1: *
 2: **
 3: ***
 4: ****
 5: *****
 6: ******
 7: *******
 8: ********
 9: *********
10: **********
```

1 から 10 までの数字を表示し、その数だけ右側に「*」を繰り返し表示するプログラムです。7 行目が、変数 j が 0 から i の間、処理を繰り返すという命令になっています。「*」を繰り返し表示する処理が終わったら、10 行目で改行を入れて、次の数字を表示する準備をします。このような入れ子構造は、二次元データや三次元データを扱う際に必ず必要になりますので、使いこなせるように練習しておきましょう。

演習 4-2 (ex4-2.c)

for 文の入れ子を利用して、九九の表を表示して下さい。

演習 4-2 表示例①

```
 1  2  3  4  5  6  7  8  9
 2  4  6  8 10 12 14 16 18
 3  6  9 12 15 18 21 24 27
 4  8 12 16 20 24 28 32 36
 5 10 15 20 25 30 35 40 45
 6 12 18 24 30 36 42 48 54
 7 14 21 28 35 42 49 56 63
 8 16 24 32 40 48 56 64 72
 9 18 27 36 45 54 63 72 81
```

演習 4-2 表示例②

```
   |  1  2  3  4  5  6  7  8  9
--------------------------------
 1 |  1  2  3  4  5  6  7  8  9
 2 |  2  4  6  8 10 12 14 16 18
 3 |  3  6  9 12 15 18 21 24 27
 4 |  4  8 12 16 20 24 28 32 36
 5 |  5 10 15 20 25 30 35 40 45
 6 |  6 12 18 24 30 36 42 48 54
 7 |  7 14 21 28 35 42 49 56 63
 8 |  8 16 24 32 40 48 56 64 72
 9 |  9 18 27 36 45 54 63 72 81
```

4.2　while 文

繰り返し処理を行う命令として、for 文の他にも while 文があります。while 文は for 文と同じようにも使うことができますが、基本的には「繰り返し回数が決まった数でないとき」によく使われます。while 文の書式は以下に示すとおりです。

```
while ( 条件式 ) {
    // 繰り返し実行する処理
}
```

for 文と比べて、大分シンプルになっていることがわかると思います。while 文では、条件式を満たす間、波括弧で囲われた処理を繰り返し実行します。ただし、繰り返し実行する処理の部分が条件式と無関係である場合、永遠に繰り返しされるので（無限ループ）注意が必要です。while 文の使用例として、次のソースコードを見てみましょう。

例 4-4 100 未満の 2 の累乗数を while 文で表示

```
1  #include <stdio.h>
2
3  int main(void) {
4      int n = 1;
5      while (n < 100) {
6          printf("%d\n", n);
7          n *= 2;
8      }
9      return 0;
10 }
```

例 4-4 実行結果

```
$ ./a
1
2
4
8
16
32
64
```

4 行目： 整数型の変数 n を宣言し、1 を代入しています。

5 行目： while 文で、n が 1 より大きい間、処理を繰り返すことを表しています。n の値は 7 行目で 2 倍になっていきます。

このプログラムは、n の値を繰り返し 2 倍にしていき、100 を超えたら終了するというものです。実は、for 文を用いて次のように書き換えることも可能です。

例 4-5 100 未満の 2 の累乗数を for 文で表示

```c
#include <stdio.h>

int main(void) {
    int n;
    for (n = 1; n < 100; n *= 2) {
        printf("%d\n", n);
    }
    return 0;
}
```

どちらが読みやすいか、書きやすいかなどを考慮して、for 文と while 文を使い分けられるようになりましょう。

演習 4-3 (ex4-3.c)

整数値の入力を繰り返し受け取り、合計値を表示するプログラムを作成して下さい(while 文を使用)。また、合計値が 100 を超えたらプログラムを終了するようにして下さい。

※予期せぬ無限ループに陥ってしまった場合は、[Ctrl] + [C]で強制終了させることができます。

演習 4-3 実行例

```
$ ./a
整数値を入力して下さい: 15
合計値: 15
整数値を入力して下さい: 60
合計値: 75
整数値を入力して下さい: 30
合計値: 105
合計値が 100 を超えたのでプログラムを終了します。
```

4.3　do～while 文

while 文とよく似た文に、do～while 文があります。while 文（および for 文）は、条件式を評価してから繰り返し処理の内容が実行されるのに対し、do～while 文では繰り返し処理の内容をまず実行してから、条件式を評価します。すなわち、while 文では処理の内容が 1 度も実行されない可能性があるのに対し、do～while 文であれば必ず 1 回は実行されます。do～while 文の書式は以下のとおりです。

```
do {
    // 繰り返し実行する処理
} while ( 条件式 );
```

do～while 文は、少なくとも 1 回は処理が実行されるという特徴から、入力の受け取り処理などで利用されることがあります。例えば、例 3-2 のソースコードを改良した以下のコードを見てみましょう。

例 4-6 while 文による例 3-2 の改良

```
1  #include <stdio.h>
2
3  int main(void) {
4      int score;
5
6      do {
7          printf("成績を入力して下さい（0～100）: ");
8          scanf("%d", &score);
9      } while (score < 0 || 100 < score);
10
11     if (score >= 60) {
12         printf("単位がもらえます。\n");
13     } else {
14         printf("不可です。\n");
15     }
16
17     return 0;
18 }
```

例 4-6 実行例

```
$ ./a
成績を入力して下さい（0～100）: -5
成績を入力して下さい（0～100）: 110
成績を入力して下さい（0～100）: 60
単位がもらえます。
```

7～8 行目で成績の入力を受け付け、9 行目の条件を満たす間、何度も繰り返し入力を促すソースコードとなっています。このループを抜け出すためには、score に 0 から 100 の数値を入れる必要があります。

演習 4-4 (ex4-4.c)

足し算の問題を表示して、正解が入力されるまで数値の入力を促すプログラムを作成して下さい（do～while 文を使用）。

※予期せぬ無限ループに陥ってしまった場合は、[Ctrl] + [C]で強制終了させることができます。

演習 4-4 実行例

```
$ ./a
9 + 19 = ?
解答を入力して下さい: 15
解答を入力して下さい: 30
解答を入力して下さい: 28
正解です。
```

4.4　break 文と continue 文

for 文と while 文による繰り返し処理をさらに使いこなすための、break 文と continue 文についても紹介します。これらは繰り返し処理の中で使われる文であり、それぞれ以下の役割を持ちます。

break 文・・・繰り返し処理を中断してループから抜ける

continue 文・・・繰り返し処理を中断してループの先頭に戻る

また、break 文と continue 文は、よく無限ループと組み合わせて利用されます。無限ループとはその名のとおり、処理を無限に繰り返すことを意味します。for 文では終了の条件式に何も書かないことで、while 文では条件式に真偽値の「真」を意味する“1”を入れることで、無限ループを作ることができます。

```
1  // for 文による無限ループ
2  for (;;) {
3      // 繰り返し実行する処理
4  }
5
6  // while 文による無限ループ
7  while (1) {
8      // 繰り返し実行する処理
9  }
```

※予期せぬ無限ループに陥ってしまった場合は、[Ctrl] + [C]で強制終了させることができます。

それでは実際に、無限ループと break 文、continue 文を組み合わせた例を見てみましょう。

例 4-7 break 文, continue 文による例 4-6 の改良

```
1  #include <stdio.h>
2
3  int main(void) {
4      int score;
5
6      while (1) {
7          printf("成績を入力して下さい（0～100）: ");
8          scanf("%d", &score);
9          if (score < 0 || 100 < score) {
```

```
10              printf("0 から 100 の範囲で入力して下さい。\n");
11              continue;
12          } else {
13              break;
14          }
15      }
16
17      if (score >= 60) {
18          printf("単位がもらえます。\n");
19      } else {
20          printf("不可です。\n");
21      }
22      return 0;
23  }
```

例 4-7 実行例

```
$ ./a
成績を入力して下さい（0〜100）: -5
0 から 100 の範囲で入力して下さい。
成績を入力して下さい（0〜100）: 110
0 から 100 の範囲で入力して下さい。
成績を入力して下さい（0〜100）: 60
単位がもらえます。
```

例 4-6 では、入力された数値が 0〜100 の範囲外であった場合、何も表示せずに再び入力を促すだけでした。一方、例 4-7 では 10 行目で「0 から 100 の範囲で入力して下さい。」と表示した後に continue 文を用いることで、よりユーザにわかりやすいプログラムとなっています。また、範囲が正しかった場合は 13 行目の break 文で、while 文の無限ループから抜け出しています。

4.5　課題

課題 4-1

正の整数 n を受け取り、for 文を使って $n!\,(=1\times2\times\ldots\times n)$ の値を表示するプログラムを作成して下さい。

課題 4-2

for 文を使って円周率を求めてみましょう。計算には、ライプニッツの公式として知られる以下の級数を用いて下さい。また、繰り返しの回数を 100 回、100 万回、10 億回と変化させて、実際の円周率との誤差を確認して下さい。

$$\pi = 4\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots+(-1)^{n-1}\frac{1}{2n-1}+\cdots\right)$$

ヒント 1：奇数項と偶数項で符号が反転しています。

ヒント 2：結果の表示が省略されないよう「`%.15lf`」等としましょう。

課題 4-3

課題 4-2 のプログラムを、while 文を使って書き直して下さい。ただし終了条件には、誤差がある値（例えば 1×10^{-9}）よりも小さくなった場合を設定して下さい。

補足：math.h の利用

本章の課題を効率よく解くために、math.h を紹介します（詳細は付録にて）。math.h とは、さまざまな数学の関数や定数が用意されたヘッダファイルであり、利用する場合は「`#include <math.h>`」のようにして読み込みます。例えば、math.h の中には円周率が定数（マクロ）として定義されており、「M_PI」と入力することで円周率の値を利用することができます。また、fabs()関数を用いることで、数値の絶対値を計算することができます。

math.h の使用例

```c
#include <stdio.h>
#include <math.h>

int main(void) {
    printf(" 3.1 - pi  = %.15lf\n", 3.1 - M_PI);        // 差の計算
    printf("|3.1 - pi| = %.15lf\n", fabs(3.1 - M_PI)); // 差の絶対値の計算
    return 0;
}
```

第 5 章　関数

まだ

第 6 章　配列とポインタ

まだ

第 7 章　構造体

まだ

第 8 章　ファイル操作

まだ

第 9 章　数値計算

まだ

第 10 章　画像処理

まだ