令和６年度

「専修学校による地域産業中核的人材養成事業」

動画教材（抜粋）

1

IT講座
基本情報技術者試験
擬似言語入門

C言語経験者のための〈擬似言語〉入門

[1]〈擬似言語〉について

はじめに(1)擬似言語 とは？

情報処理技術者試験の〈擬似言語〉とは？

受験者が経験したことのあるプログラム言語は千差万別。特定の言語に寄せた出題をしては不公平が生じます。なぜなら，基本情報技術者試験で測りたい力は，特定の言語の知識ではなく，任意の言語を用いて設計仕様どおりのロジックを実装する力だからです。

よって，本来，受験者全員が，その試験のためだけに毎回新たに作られたプログラム言語の仕様をその場で初めて読んで設問に答えるのが最も公平なはずですが，現実の試験の形式として，それは不可能に近いでしょう。

そこで，誰も実際に開発に用いたことのない試験専用の言語仕様を定義し，それに基いて試験問題を作成することで，一定の公平性を担保するためのプログラム言語が〈擬似言語〉です。

実際のITエンジニアの仕事でも，自分にとって未知の言語の概要を短期間で学習し，少なくともその言語の特性を理解しなくてはならないケースはあります。未知の言語の仕様を読んで，その言語がおおまかに既知のどの言語に似ているのかを掴む力は，培っておいて損はないでしょう。

1

はじめに(2)擬似言語 へのアプローチ

実在するプログラム言語に触れたことがない人が，基本情報技術者試験を受験するためだけに〈擬似言語〉を学習するのは，あまり意味がありません。なぜなら，〈擬似言語〉はあくまで試験のためだけに存在する言語なので，“〈擬似言語〉しか知らない技術者”になったところで，現実の開発技術者としてはニーズがないのです。

しかし，なんらかのプログラム言語に触れたことのある人が〈擬似言語〉を学習することには，意味があります。日本語や英語などの自然言語についても言えることですが，ひとつしか言語を知らない人は，その言語の特徴を知っているとは言えません。ほかの言語と比べてはじめて，最初に知った言語の特徴を意識することができるのです。

この教材を利用する方々は，学校の授業などで多かれ少なかれC言語の経験があるはずなので，ここではC言語を足がかりに〈擬似言語〉を学習するというアプローチを取ります。それは，とりもなおさず，C言語をより深く知ることにも繋がります。

C言語経験者のための〈擬似言語〉入門
[1]〈擬似言語〉について

1

この教材で学ぶ 擬似言語 の記述と意味

宣言部

- 手続き，または関数の宣言
- 変数の宣言
- 配列の宣言
- 注釈の記述

処理部

- 変数に式の値を代入(順次処理)
- 演算子と優先順位
- 手続き，または関数の呼び出し，引数の受け渡し
- 選択処理
- 繰り返し処理

その他，基本的なルール

- 未定義，未定義の値

1

擬似言語 について知っておくべき前提

➡ 擬似言語は走りません

〈擬似言語〉で書いたプログラムは，“走らせる”こと，つまり，動かすことはできません。言語と言ってもその仕様は，実際のプログラム言語に比べればずっとおおざっぱで，動くプログラムが開発できるようなものではないのです。“流れ図”で示される概念や手続きを，あたかもプログラムのように書き表すための簡易言語と捉えておけばよいでしょう。

➡ 擬似言語には，決まっていないことが数多くあります

既存のプログラム言語の概念や記述に対応するものが，必ず〈擬似言語〉に用意されているとはかぎりません。〈擬似言語〉のほうがはるかに簡略化されているので，憶えなければならないことはさほど多くはありません。

➡ 出題では，モジュールに分割されていることが前提となっています

〈擬似言語〉での出題では，長大な1本のプログラムではなく，小さなモジュールに分割されたプログラムが，他のモジュールから呼び出されることが前提になっています（「モジュール」とは，ある程度まとまった機能を持つ“部品”のようなものだと思っておいてください）。したがって，モジュール間で受け渡されるデータ（引数や戻り値）を意識する必要があります。「このモジュールは，なにをしようとしているのか」を掴めれば，細部の設問の答えも見えてくるでしょう。

2

IT講座
基本情報技術者試験
擬似言語入門

C言語経験者のための〈擬似言語〉入門

[2]宣言部(1)

宣言部と処理部とは？

- 〈擬似言語〉のプログラムは，多くのプログラム言語がそうであるように，「宣言部」と「処理部」から成ります。
- 「宣言部」では，そのプログラムの名前(「手続名」や「関数名」)や「処理部」で用いる変数や配列などを定義します。
- 「処理部」では，「宣言部」で定義された変数や配列などを用いて，プログラムを動作させる命令を記述してゆきます。

C言語での例

```
[宣言部]
int price, includingTax = 0;
int tax = 0.1;


[処理部]
includingTax = price + price * tax;
```

2

宣言部 手続を宣言する（戻り値・引数なし）

※「戻り値(もどりち)」は「返り値(かえりち)」とも呼びます。

擬似言語の記述形式

【戻り値も引数もない場合の形】

- 手続名()

例1擬

- show_date()

C言語の記述形式

【戻り値も引数もない場合の形】

```
void 手続名 (void);
```

例1C

```
void show_date(void);
```

プログラム名(手続名)に戻り値も引数もない場合は，[例1擬]のようにプログラム名に「データ型」を指定せず，空の()を書きます。C言語の場合は，[例1C]のように，戻り値や引数がないことを，「void」を記すことで明示します。

2

宣言部 手続または関数を宣言する（戻り値・引数あり）

擬似言語の記述形式

【基本形】

- 手続名または関数名

【戻り値や引数がある場合の形】

- 戻り値のデータ型:手続名または関数名
  (引数のデータ型:引数名, …)

例2擬

- 整数型:includingTax(整数型:price)

C言語の記述形式

【基本形】

対応する形式なし

【戻り値や引数がある場合の形】

```
戻り値のデータ型 手続名または関数名
(引数のデータ型 引数の名前, …);
```

例2C

```
int includingTax(int price);
```

プログラム名(手続名または関数名)includingTax に**戻り値**がある場合は，[例2擬]のようにプログラム名に「**データ型**」を指定します。擬似言語ではデータ型は日本語で，C言語では「int」「char」など主に英語を元に決められた表記を用います。データ型の種類と名称(型名)については，「変数を宣言する」のところで詳しく述べます。

3

IT講座
基本情報技術者試験
擬似言語入門

C言語経験者のための〈擬似言語〉入門

[3]宣言部(2)

宣言部 変数を宣言する(1)

擬似言語の記述形式

【基本形】
　型名:変数名

例3擬
　整数型:price

C言語の記述形式

【基本形】
　型名 変数名;

例3C
　int price;

擬似言語のデータ型は，C言語などの実際のプログラム言語に比べるとはるかに簡略化されていて，基本的な型は5種類しかありません。C言語にはあるが，擬似言語にはないデータ型も多く，また，両言語のデータ型の意味するところも，厳密に同じではありません。

3

宣言部 変数を宣言する(2)

擬似言語の変数のデータ型とC言語のデータ型の対比

※厳密にはまったく同じ意味ではなく，最も近いものを挙げている。

変数のデータ型		説明
擬似言語	C言語	
整数型	int	整数を表す
実数型	double	浮動小数点数を表す
文字型	char	1文字を表す
文字列型	char 配列	文字の並びを表す
論理型	bool	真(true)または偽(false)を表す

【C言語での「論理型」について】

じつは，古いC言語には「論理型」がそもそも存在せず，「整数型」の変数に，論理値が「真」なら「1」，「偽」なら「0」を格納することで代用していました。C99以降のC言語では，標準ライブラリstdbool.h がサポートされ，「true」「false」の論理型が使えるようになりました。

3

宣言部 変数を宣言する(3)

擬似言語の記述形式

【複数の変数をまとめて宣言】
型名:変数名1, 変数名2

例4擬 ※同じ型の変数は複数をまとめて宣言できる
整数型:price, setPrice

例5擬 ※異なる型の変数は別々に宣言する
整数型:price, setPrice
文字型:code_01

C言語の記述形式

【複数の変数をまとめて宣言】
型名 変数名1, 変数名2;

例4C
int price, setPrice;

例5C
int price, setPrice;
char code_01;

擬似言語でもC言語でも，宣言する変数のデータ型が同じ場合は，まとめて宣言することができます。
型が異なる変数は，別々に宣言します。このルールは，擬似言語でもC言語でも同じです。

3

宣言部 変数を宣言する(4)

擬似言語の記述形式

【変数の宣言時に初期値を代入する】
　型名:変数名 ← 初期値

例6擬

```
整数型:value ← 0
```

C言語の記述形式

【変数の宣言時に初期値を代入する】
　型名 変数名 = 初期値;

例6C

```
int value = 0;
```

擬似言語でもC言語でも，変数を宣言すると同時に，初期値を代入する(**初期化する**)ことができます。
　C言語では，「=」は**代入演算子**と呼び，式の右辺の値を左辺の変数に代入することを表します。式の左辺と右辺の値が等しいかどうかをチェックするときは「==」(**比較演算子**)を用います。
　「=」は，さまざまなプログラム言語で，数学の等号と同じ文字を用いていても意味が異なることが多く，新しい言語を学習するときには，とくに要注意です。代入演算子に「<-」や「:=」などを用いる言語もあります。

4

IT講座
基本情報技術者試験
擬似言語入門

C言語経験者のための〈擬似言語〉入門

[4]宣言部(3)

宣言部 配列を宣言する(1)

擬似言語の記述形式

【基本形】
　データ型名の配列:配列名

例7擬
　整数型の配列:height

C言語の記述形式

【基本形】
　型名 配列名[要素数];

例7C
　int height[10];

擬似言語の配列の大きな特徴は，配列の要素数を定義せずに宣言できる点です。C言語では，配列を宣言する際には，要素がいくつある配列なのか，必ず「要素数」を併せて宣言します。

4

宣言部 配列を宣言する(2)

擬似言語の記述形式

【配列を宣言し，初期値を設定する】
整数型の配列:配列名 ← {値$_1$, 値$_2$, 値$_3$,…… 値$_n$}

例8擬
整数型の配列:pn ← {2,3,5,7,11,13,17,19,23,29}

C言語の記述形式

【配列を宣言し，初期値を設定する】
int 配列名[ ] = {値$_1$, 値$_2$, 値$_3$,…… 値$_n$};
または
int 配列名[n] = {値$_1$, 値$_2$, 値$_3$,…… 値$_n$};

例8C
int pn[ ] = {2,3,5,7,11,13,17,19,23,29};
または
int pn[10] = {2,3,5,7,11,13,17,19,23,29};

擬似言語もC言語も，配列の宣言と同時に初期値を設定することができます。前述のように，擬似言語では配列を宣言する際に要素数を記述しませんが，C言語では，配列の宣言と同時に初期値を設定するときのみ，要素数を省略することができます。ただし，要素数を省略するときにも，[ ] は記述する必要があります。

4

宣言部 配列を宣言する(3)

注意！ **配列の要素番号について**

擬似言語では，配列の要素番号について，とくに気をつけておくべきことがあります。
たとえば，[例9擬]のように，擬似言語で配列を宣言し，値を設定したとします。

例9擬
整数型の配列:numbers ← {1,2,3,4,5,6,7,8,9,10}

さて，配列の先頭の値「1」が格納されている領域を指す記述は，numbers[0] でしょうか，numbers[1] でしょうか？

じつは，擬似言語には，配列の先頭を指す要素番号の取り決めがありません。問題によって [0] であるときと [1] であるときがあります。
実際のプログラム言語では，C言語や Python など，配列の要素番号が [0] からはじまる言語が主流ではありますが，歴史のある FORTRAN のように，とくにユーザが設定しなければ，[1]からはじまる言語も少数ながら存在します。
基本情報技術者試験の擬似言語の場合は，要素番号の先頭を勝手にどちらと思い込まずに，その都度，問題をよく読んで見きわめましょう。
みなさんは，これからさまざまなまだ見ぬプログラム言語とつきあってゆくことになるでしょうから，試験ではそうした柔軟性も問われているのです。

4

宣言部 配列を宣言する(4)

擬似言語の記述形式

【二次元配列の宣言】
　データ型名の二次元配列:配列名

例10擬

　整数型の二次元配列:examScore

※擬似言語の場合は，二次元配列として宣言するだけで，何行・何列なのかは，値を代入するまでは決まりません。

C言語の記述形式

【二次元配列の宣言】
　データ型名 配列名[行数][列数];

例10C

```
int examScore[3][5];
```

※C言語なら，たとえば，こういうイメージです。値は「未定義」の3行・5列の二次元配列が宣言されています。

	[0]	[1]	[2]	[3]	[4]
[0]	(未定義)	(未定義)	(未定義)	(未定義)	(未定義)
[1]	(未定義)	(未定義)	(未定義)	(未定義)	(未定義)
[2]	(未定義)	(未定義)	(未定義)	(未定義)	(未定義)

4

宣言部 配列を宣言する(5)

擬似言語の記述形式

【二次元配列の宣言と同時に値を代入する】
　整数型の二次元配列:配列名 ← {{値11, 値12, 値13, ……}, {値21, 値22, 値23, ……}, {値31, 値32, 値33, ……}, ……}

例10擬

　整数型の二次元配列:examScore ←{{83, 68, 79, 98, 54}, {92, 45, 77, 95, 82}, {34, 76, 100, 84, 66}}

	[1]	[2]	[3]	[4]	[5]
[1]	83	68	79	98	54
[2]	92	45	77	95	82
[3]	34	76	100	84	66

※要素番号は先頭を[1]とする。

C言語の記述形式

【二次元配列の宣言と同時に値を代入する】
　int 配列名[行数][列数] = {{値11, 値12, 値13, ……}, {値21, 値22, 値23, ……}, {値31, 値32, 値33, ……}, ……}

例10C

```
int examScore[][5]= {{83, 68, 79, 98, 54},
{92, 45, 77, 95, 82}, {34, 76, 100, 84, 66}};
```

C言語では，宣言と同時に値を代入する場合のみ，行数を省略することができます。

宣言部 配列を宣言する(6)

前ページで宣言した二次元配列の要素を表現する方法には，とくに注意しましょう。

例10擬

整数型の二次元配列:examScore ← {{83, 68, 79, 98, 54},
{92, 45, 77, 95, 82}, {34, 76, 100, 84, 66}}

	[1]	[2]	[3]	[4]	[5]
[1]	83	68	79	98	54
[2]	92	45	**77**	95	82
[3]	34	76	100	84	66

※要素番号は先頭を[1]とする。

例10C

```
int examScore[][5]= {{83, 68, 79, 98, 54},
 {92, 45, 77, 95, 82}, {34, 76, 100, 84, 66}};
```

	[0]	[1]	[2]	[3]	[45]
[0]	83	68	79	98	54
[1]	92	45	**77**	95	82
[2]	34	76	100	84	66

擬似言語では，二次元配列の要素を**[2,3]**のようにカンマで区切って表現します。[2,3]は「**2行めの3列め**」という意味です。上の擬似言語の例では **examScore[2,3]** に代入されている値は「**77**」です。
一方，C言語では，配列の先頭の要素番号は必ず[0]ですので，上記の「77」が代入されている要素は，**examScore[1][2]** と表されます。
「配列を宣言する(3)」でも述べたように，擬似言語では配列の先頭の要素番号に関する取り決めがなく，問題によって変わります。よく問題を読んで，けっして早合点しないようにしましょう。

4
C言語経験者のための〈擬似言語〉入門
[4]宣言部(3)

注釈(コメント) を記述する ※注釈の記述は「宣言」ではありませんが，ここで触れておきます。

擬似言語の記述形式

【注釈の記述】

```
/* 注釈 */
   または
//  注釈
```

例11擬 文中，または複数行にわたる注釈

```
整数型の配列:pn ← {2,3,5,7,11,13,17,19,23,29}
   /* 整数型の配列 pn を宣言し,
      最初の素数10個を代入する */
```

例12擬 1行分に対する注釈

```
整数型の配列:pn ← {2,3,5,7,11,13,17,19,23,29}
  // 配列を宣言し，最初の素数10個を代入
```

C言語の記述形式

【注釈の記述】

```
/* 注釈 */
   または
//  注釈
```

例11C 文中，または複数行にわたる注釈

```
int pn[ ] = {2,3,5,7,11,13,17,19,23,29};
/* 整数型の配列 pn を宣言し,
   最初の素数10個を代入する */
```

例12C 1行分に対する注釈

```
int pn[ ] = {2,3,5,7,11,13,17,19,23,29};
 //配列を宣言し，最初の素数10個を代入
```

5

IT講座
基本情報技術者試験
擬似言語入門

C言語経験者のための〈擬似言語〉入門

[5]処理部(1)

処理部 変数や配列に式の値を代入する(順次処理)(1)

擬似言語の記述形式

【基本形】
　変数名 ← 式

例13擬

```
a ← 0        /* 変数aに0を代入 */
deadline ← shimekiri
/* 変数deadlineに変数shimekiriの値を代入 */
```

C言語の記述形式

【基本形】
　変数名 = 式;

例13C

```
a = 0;     /* 変数aに0を代入 */
deadline = shimekiri;
/* 変数deadlineに変数shimekiriの値を代入 */
```

上記の【基本形】にある「式」は，**数学で言う「式」とは少し意味が異なる，プログラム言語での言葉遣いであることに注意**してください。数学の「式」は，式を構成する要素のあいだの「関係そのものを表現する」ことに重点がありますが，プログラム言語での「式」は，あくまで「実行」の一部であり，動作や実行結果に重点を置いています。[例13擬]と[例13C]の「0」や「shimekiri」は，**ひとつの要素だけで構成されていても，いずれも「式」と呼びます。**
宣言済みの変数に数値や，他の変数の値を代入する処理の記述は，擬似言語もC言語も基本的な形はさほど変わりません。C言語の「=」は，数学の等号ではなく，代入演算子だということを思い出しておいてください。

処理部 変数や配列に式の値を代入する(順次処理)(2)

擬似言語の記述形式

【配列の要素への値の代入】
配列名[要素番号] ← 式

例14擬

```
price[1] ← 0
/* 配列priceの要素番号1の要素に0を代入 */

height_weight[2,6] ← 0
/* 二次元配列height_weightの要素番号[2,6]
の要素に0を代入 */
```

C言語の記述形式

【配列の要素への値の代入】
配列名[要素番号] = 式;

例14C

```
price[0] = 0;
/* 配列priceの要素番号0の要素に0を代入 */

height_weight[2][6] = 0;
/* 二次元配列height_weightの要素番号[2][6]
の要素に0を代入 */
```

擬似言語の配列の要素番号については,『[宣言部]配列を宣言する(3)』をふり返っておきましょう。擬似言語では, [0]からはじまる場合もあれば, [1]からはじまる場合もあります。

5
C言語経験者のための〈擬似言語〉入門
[5]処理部(1)

処理部 変数や配列に式の値を代入する(順次処理)(3)

擬似言語の記述形式

【変数に演算結果の値を代入】
変数名 ← 式

例15擬

```
(1) r ← p × q
    /* 変数 p の値と変数 q の値の積を変数 r に代入 */

(2) a ← (b mod 7)
    /* 変数 b の値を7で割った剰余を変数 a に代入 */

(3) total ← total + subtotal
    /* 総合計 total に小計 subtotal を加算する */
```

C言語の記述形式

【変数に演算結果の値を代入】
変数 = 式;

例15C

```
(1) r = p * q;
    /* 変数 p の値と変数 q の値の積を変数 r に代入 */

(2) a = b % 7;
    /* 変数 b の値を7で割った剰余を変数 a に代入 */

(3) total = total + subtotal;
          または
    total += subtotal;
    /* 総合計 total に小計 subtotal を加算する */
```

処理部 変数や配列に式の値を代入する（順次処理）（4）

［擬似言語の演算子と優先順位］

演算子の種類		演算子	優先度
式		( ) .	高
単項演算子		not ＋ －	
二項演算子	乗除	mod × ÷	
	加減	＋ －	
	関係	≠ ≦ ≧ ＜ ＝ ＞	
	論理積	and	
	論理和	or	低

注記
演算子 . は，メンバ変数又はメソッドのアクセスを表す。
演算子 mod は，剰余算を表す。

6

IT講座
基本情報技術者試験
擬似言語入門

C言語経験者のための〈擬似言語〉入門

[6]処理部(2)

処理部 手続または関数を呼び出し, 引数を受け渡す

擬似言語の記述形式

【基本形】
手続名または関数名(引数, …)

例16擬
```
result ← sum(3, 7)
```

C言語の記述形式

【基本形】
手続名または関数名(引数, …);

例16C
```
result = sum(3, 7);
```

手続または関数を呼び出して引数を受け渡す命令は, 擬似言語でもC言語でも, ほぼ同じです。上記の例では, sum という名前の関数に, 引数 3 と 7 を受け渡しています。たとえば, sum関数がふたつの整数の引数の合計値を戻り値として返す関数だった場合, sum関数からの戻り値を代入する先のresult という変数は, 宣言部で「整数型」として適切に宣言されていなくてはなりません。

処理部 選択処理(1)

擬似言語の記述形式

【基本形】

```
if ( 条件式1 )
    処理1
  elseif ( 条件式2 )
    処理2
  elseif ( 条件式n )
    処理n
  else
    処理n+1
  endif
```

C言語の記述形式

【基本形】

```
if (条件式1) {
  // 処理1
} else if (条件式2) {
  // 処理2
} else if (条件式n) {
  // 処理n
} else {
  // 処理n+1
}
```

〈擬似言語〉唯一の選択処理のための記述 if は，非常にシンプルです。条件式を**上から順に評価し，最初に真になった条件式に対応する処理を実行**します。以降の条件式は評価せず，実行もしません。
どの条件式も真にならないときは，処理n+1 を実行します。「もし～だったら(if)」「さもなければ，もし～だったら(elseif)」「これらのもしのどれでもなかったら(else)」と考えるとわかりやすいでしょう。

処理部 選択処理(2)

擬似言語の記述形式

【処理をするかしないかに分ける】

例17擬

```
if (購入合計額が3500円未満)
    total ← total + 800  // 購入合計額に送料を加える
endif
```

C言語の記述形式

【処理をするかしないかに分ける】

例17C

```
if (total < 3500) {
  total = total + 800;  // 購入合計額に送料を加える
}
                または
if (total < 3500) {
  total += 800;  // 購入合計額に送料を加える
}
```

最も単純な選択処理です。ある条件が成り立つか成り立たないかで，処理をするかしないかに分けます。

処理部 選択処理(3)

擬似言語の記述形式

【2通りの処理に分ける】

例18擬

```
if (購入合計額が3500円未満)
    total ← total × 0.9  // 購入合計額から10%割り引く
else
    total ← total × 0.8  // 購入合計額から20%割り引く
endif
total ← total + 800  // 購入合計額に送料を加える
```

C言語の記述形式

【2通りの処理に分ける】

例18C

```
if (total < 3500) {
    total = total * 0.9;  // 購入合計額から10%割り引く
} else {
    total = total * 0.8;  // 購入合計額から20%割り引く
}
total += 800;  // 購入合計額に送料を加える
```

上記のプログラムでは，購入合計額が3500円未満か否かによって，割引金額を変える処理が行われています。送料は，割引金額にかかわらず，一律で800円かかるという処理になっていますので，送料を加算する処理は，〈擬似言語〉では，if – else – endif で囲まれた部分の外に書かれています。もし，割引額が変われば送料も変わるのであれば，〈擬似言語〉のほうのプログラムには，else の前に1行，endif の前に1行，異なる送料を加算する処理を書き加えて，最後の行は削除せねばなりませんね。
C言語のほうも，条件によって送料も変わるように書き換えてみてください。

処理部 選択処理(4)

擬似言語の記述形式

【3通り以上の処理に分ける】

例18擬

```
if (購入合計額が5000円以上)
    total ← total × 0.7  // 購入合計額から30%割り引く
elseif (購入合計額が3500円以上)
    total ← total - 700  // 購入合計額から700円割り引く
else
    total ← total -350  // 購入合計額から350円割り引く
endif
```

C言語の記述形式

【3通り以上の処理に分ける】

例18C

```
if (total >= 5000) {
    total = total * 0.7; // 購入合計額から30%割り引く
} else if (total >= 3500) {
    total = total - 700; // 購入合計額から700円割り引く
} else {
    total = total - 350; // 購入合計額から350円割り引く
}
```

3通り以上の処理に分ける場合も，〈擬似言語〉なら elseif と条件式，C言語なら else if と条件式が増えてゆくだけで，大枠のカタチは同じです。〈擬似言語〉では elseif と1語で，C言語では else if と2語に分かち書きする点に気をつけましょう。

7

IT講座
基本情報技術者試験
擬似言語入門

C言語経験者のための〈擬似言語〉入門

［7］処理部（3）

処理部 繰返し処理

繰返し処理は，選択処理と順次処理を組み合わせた処理と言えます。変数の値を変化させながら，条件文に記述された条件を満たさなくなるまで，処理を繰返します。

したがって，繰返し処理には，以下の3つが必ず必要となります。

（1）初期値の設定

（2）繰返しの終了条件の判定

（3）累計処理および制御変数の更新

上記（2）の「繰返しの終了条件の判定」のタイミングや方法によって，繰返し処理は「前判定繰返し処理」，「後判定繰返し処理」「制御記述を使う繰り返し処理」の3種類に分類されます。以下，ひとつずつ見てゆきましょう。

処理部 繰返し処理ー前判定繰返し処理

擬似言語の記述形式

【基本形】

```
while（条件式 ※継続条件の形で記述 ）
    処理
endwhile
```

例20擬

```
total ← 0  // 累計した値を入れる変数の初期化
i ← 1  // 制御変数に初期値を設定
while（i ≦ 2000）
  total ← total + i  // 値を累計
  i ← i + 1    // 制御変数を更新
endwhile
```

C言語の記述形式

【基本形】

```
while（条件式 ※継続条件の形で記述 ）{
      // 処理
}
```

例20C

```
int total = 0; // 累計した値を入れる変数の初期化
    int i = 1;     // 制御変数に初期値を設定
while（i <= 2000）{ // 条件: i が2000以下のあいだ,
ループを実行
   total = total + i; // 値を累計
     i = i + 1;        // 制御変数を更新
 }
```

前判定繰返し処理は，ループの処理に入る前に条件の判定を行います。ここで〈擬似言語〉のルールとして非常に重要なのは，条件式の部分には，**“ループを終了させる条件”（終了条件）ではなく，“ループを継続させる条件”（継続条件）を記述する**ことです。「i が2000を超えたらループを終える」と「i が2000以下のあいだはループを続ける」は，意味はまったく同じですが，**〈擬似言語〉では，必ず「継続条件」の形式で条件を記述する**ことになっています。じつは，これはC言語の while も同じです。ほかのプログラム言語には，「指定した条件が満たされるまで繰り返し処理を実行する」という意味で「終了条件」を記述する命令 until などがあるので，参考までに知っておきましょう。

処理部 繰返し処理ー後判定繰返し処理

擬似言語の記述形式

【基本形】

```
do
    処理
while（条件式 ※継続条件の形で記述 ）
```

例21擬

```
  total ← 0  // 累計した値を入れる変数の初期化
  i ← 1  // 制御変数に初期値を設定
do
  total ← total + i  // 値を累計
  i ← i + 1  // 制御変数を更新
while（i ≦ 2000）
```

C言語の記述形式

【基本形】

```
do {
      // 処理
} while（条件式 ※継続条件の形で記述 ）;
```

例21C

```
int total = 0;  // 累計した値を入れるの初期化
int i = 1;  // 制御変数に初期値を設定
do {
      // 処理
} while (i ≦ 2000);
```

後（あと）判定繰返し処理は，**ループの中に記述された処理を行なってから条件の判定を行います。**つまり，**必ず一度はループの中の処理が実行される**点が，前判定繰返し処理と大きく異なります。前判定繰返し処理の場合は，条件判定の結果，ループに入らず，処理が一度も行われないこともあり得ます。

処理部 繰返し処理ー制御記述を使う繰返し処理

擬似言語の記述形式

【基本形】

```
for ( 制御記述 )
    処理
endfor
```

※「制御記述」には，「どの変数をいくつからいくつまで，いくつずつ増やす」という形式の記述をする。［例22擬］と，右のC言語での記述を参照。

例22擬

```
total ← 0  // 累計する値を入れる変数を初期化
for ( i を1から2000まで 1 ずつ増やす )
    total ← total + i  // 値を累計
endfor
```

C言語の記述形式

【基本形】

```
for (初期化式; 条件式; 更新式) {
        // 処理
}
```

例22C

```
int total = 0;  // 累計する値を初期化
int i;  // ループ制御変数を宣言
for (i = 1; i <= 2000; i++) { // i を1から2000まで
1ずつ増やす
        total = total + i;  // 値を累計
}
```

制御記述を使う繰返し処理は，前判定繰返し処理や後判定繰返し処理では3か所に分けて記述していた繰返し処理の制御を，ループの最初の for（制御記述）の部分にすっきりとまとめて書くことができるため，論理が読み取りやすいという利点があります。

8

IT講座
基本情報技術者試験
擬似言語入門

C言語経験者のための〈擬似言語〉入門

[8]その他のルール

その他のルール 未定義，未定義の値(1)

これは問題用紙に書かれている〈擬似言語〉の記述形式の中でも，とくにわかりにくいものでしょう。「擬似言語の記述形式(基本情報技術者試験，応用情報技術者試験)」に書かれている[未定義，未定義の値]の説明を，いま一度詳しく見ておきましょう。

未定義，未定義の値

変数に値が格納されていない状態を，“未定義”という。変数に“未定義の値”を代入すると，その変数は未定義になる。

ちょっと読んだだけでは，なにを言っているのかさっぱりわかりません。変数に値が格納されていない状態が“未定義”だと言った舌の根も乾かぬうちに，「“未定義の値”を代入する」とは，はてさて，禅問答のようです。

たとえば，「整数型：price」と，変数を宣言したとします。変数が宣言されただけの状態では，price は整数型の変数だということがわかっているだけで，そこにはまだ「値」は格納されていません。「price ← 0」のように値を代入してやってはじめて，price の値が決まります。

では，0 が格納された変数 price を，もう一度，変数の宣言直後の“未定義”の状態に戻すには，どうすればよいでしょうか？　そこで用いるのが“未定義の値”という抽象的な概念です。「変数に代入することによって変数が宣言直後の“未定義”の状態になるような“値”」を，“未定義の値”と呼んでいるわけです。“未定義の値”は“値”ですから，変数 a に“未定義の値”を代入し，変数 b に“未定義の値”を代入してやれば，a と b の値は“等しい”と言えます。これは〈擬似言語〉の抽象的な決めごとですので，そういうものなのだと受け容れてください。

その他のルール 未定義，未定義の値（2）

さて，〈擬似言語〉では，変数 a に“未定義の値”が格納されており，変数 b に“未定義の値”が格納されていれば，a と b の値は“等しい”のでした。しかしこれは，コンピュータ上で実際に動作することを想定されていない〈擬似言語〉での抽象的な決めごとであって，実際にコンピュータというハードウェア上で動くプログラム言語では，まったく事情が異なります。

たとえば，C言語で「int price;」と変数を宣言したとしましょう。宣言しただけですから，変数 price は「未定義」の状態です。

しかし，このC言語がコンピュータ上で動くとき，変数 price の宣言によって確保されたメモリ上の領域には，「未定義」ではあっても，なんらかの値が入っている可能性があります。このプログラムの前に動いたプログラムによって格納されたままになっている，メモリ上の“ごみ”のような値が入っていることも多いです。ですから，実際のプログラム言語の場合は，「変数 a が“未定義”の状態で，変数 b も“未定義”の状態であるとき，変数 a と b の値は等しい」とは言えません。変数が“未定義”の状態のとき，その“値”は“不定”です。プログラムを実行するたびに異なる可能性もあります。

このように，〈擬似言語〉と，C言語などの実際にコンピュータ上で動かすためのプログラム言語とは，その抽象度がまるで異なります。〈擬似言語〉は，ハードウェアのことをほとんど考慮せずに論理だけを考えることができる抽象的なルールの集まりです。しかし，実際のプログラム言語では，そのプログラムの動作環境なども考慮に入れて，プログラムを開発する必要があります。

プログラムを数学の数式のようなものだと考えていると，思わぬところでつまずくことがあります。「おかしいな？　論理的には正しいのに，なぜかここで異常終了してしまう」などという現象に遭遇することも少なくありません。実際にコンピュータというハードウェアの上で動くプログラムは，純粋に論理だけでできている数学の数式に比べると，ずっと泥臭い，物理的制約に左右されたりするものだということを，〈擬似言語〉とC言語の比較を通じて，肝に銘じておきましょう。

おわりに

基本情報技術者試験，応用情報技術者試験の問題文に記されている「擬似言語の記述形式」は，ほんのA4用紙2ページほどのシンプルな内容です。これが〈擬似言語〉の公式な言語仕様です。これをすべて憶えるのには，さほどの労力は要しません。

しかし，〈擬似言語〉を用いた問題では，個別の問題の中で，この公式の言語仕様には述べられていない新たなルールが示されることがあります。そのような新たなルールにも臨機応変に対処できる能力が試験では求められていると言えましょう。

「[はじめに（2）]〈擬似言語〉へのアプローチ」でも述べたように，この教材はC言語に触れたことのある人を対象にしていますが，〈擬似言語〉はあくまで既存の言語を簡略化したものですから，C言語が持っているすべての記述や概念と1対1に対応する記述や概念が〈擬似言語〉にもあるわけではありません。〈擬似言語〉の内容をC言語で記述した説明についても，その意味するところが，厳密にまったく同じというわけではありません。このあたりは，われわれが日常的に使っている自然言語も同じですね。日本語の単語とぴったり1対1に対応する単語が，外国語にあるとはかぎりません。逆のこともまた言えます。

この教材の目的は，〈擬似言語〉という，ふだんあまり見かけない情報処理技術者試験のためだけの言語に，少しでも慣れてもらうことです。この教材で，〈擬似言語〉の“ノリ”がなんとなく掴めたら，それだけで合格に一歩近づくのではないかと思います。