0. はじめに

この文書は、引き継ぎのために執筆されたものである。
筆者は、この文書の正確性およびその他の一切を保障しない。
この文書は、一部分である。

1. ネットワーク

1.1. ネットワークの構成

ネットワーク全般を比較的広い意味で「インターネット」の語を使うこともあるが、ここでは全世界に接続されたあるネットワークを指して狭い意味で (固有名詞的に) 用いている。

1.1.1.IP アドレス

ネットワーク上の端末を識別するために、IP アドレスというものが存在する。IP アドレスは時に住所に喩えられる。IPv4 の IP アドレスの例を次に示す :

```
210.172.160.130
```

IP アドレスは、グローバル IP アドレスとローカル IP アドレス (プライベート IP アドレス) に分けられる (この他に、特殊な IP アドレスが存在する)。

グローバル IP アドレスはインターネット上で参照可能なアドレスであり、したがって、グローバル IP アドレスは全世界でユニーク (すなわち、グローバル IP アドレス 1 つに対して必ず 1 つのネットワークまたは機器が対応する) である。

ローカル IP アドレスはローカルな (インターネットとの間に「隔絶」のある) ネットワークで用いられる IP アドレスである。"192.168." で始まる IP アドレスはローカル IP アドレスである。

1.2. 通信

ここでは、ネットワーク上で通信を行うためのプロトコルのうち、HTTP を念頭に置いて説明する。

1.2.1. プロトコル

インターネットで通信を行うときは、一定の形式に基づいてデータを送受信する。この一定の形式のことを、「プロトコル」と呼ぶ。

現在一般的に用いられているプロトコルは多数あるが、これらはいくつかの“層”(“layer”) に分類できる。層が下位であるほど、物理的なプロトコルである。ここでは代表的なプロトコルを下位のプロトコルから順に紹介する。

- 物理層・リンク層
  イーサネットや Wi-Fi などはリンク層に位置付けられる。これらのレイヤーはネットワークを物理的に構成し、通信を可能にする
- TCP/IP
  厳密にはこれらは TCP と IP に分けることができる。これらのプロトコルにより、データ (バイナリデータ、すなわち 2 進の数値列。ASCII によって表された文字列と捉えることもできる) を転送する。
- アプリケーション層
  HTTP, FTP などのプロトコルである。これらは TCP/IP を利用してファイ

ルなどの意味を持つデータを転送する。

1.2.2. クライアントとサーバー

HTTP, FTP などの (ある意味で古典的な) プロトコルは、2 台の機器の間で通信する際、これらの機器は一方が「クライアント」、一方が「サーバー」であるという関係を持つ。

こういった (ある意味で) 単純な通信は、「クライアントがリクエストを送信→サーバーが応答 (レスポンスを返す)」という手順によって行われる。

1.2.3. ポート

通信を行う際、相手の情報は (グローバル)IP アドレスとポート番号によって示される。ポート番号は、その通信に対してどのアプリケーションが対応すべきかということを示している。例えば HTTP の場合は通常ポート番号 80 を用いる。ポート番号 80 のリクエストを受けた機器は HTTP サーバープログラムがそのリクエストに対応し処理を行う。

また、インターネットに対してルーターが接続されているような場合、1 つのグローバル IP アドレスに対して複数の機器が対応することになる。この場合、機器の振り分けもポート番号によって行われることになる。(見方を変えると、ローカルネットワークの中で機器が役割分担していると捉えることもできる)

なお、IP アドレスとポート番号を合わせて次のような表記を用いることがある :

```
210.172.160.130:80
```

2. プログラミング

この文書では、手続き型プログラミングについてのみ説明する。

手続き型プログラミングでは、プログラマは命令を記述し、コンピューターは (プログラムは) 命令を (順に) 実行する。

2.1. 語の説明

まず、いくつかの単語の意味を説明する。

2.1.1. 変数

変数は、時に箱に喩えられる。変数は、値を保持する。(「値」は、何らかのデータである。言語によっても異なるが、数値・文字列・配列などさまざまである。)

変数の (変数が保持している) 値を変更することを、「変数に値を代入する」という。

2.1.2. リテラル

リテラルは、値を直接プログラムに書く表現である。リテラルの例をいくつか示す (言語によって異なる):

```
5
"text"
[1, 1, 2, 3, 5, 8]
["foo", "bar"]
```

2.1.3. 型 (type)

「型」は、データの種類である。数値型・文字列型などがある。言語によっては、変数と変数に代入する値の型が一致している必要があるなどの制約がある。

2.1.2.1. 真偽型 (bool 型・boolean 型)

重要な型として真偽型というものがある。この型は真 (true)・偽のどちらかの値を取るものである。

2.1.2.2. 型のキャスト

2 つ以上の値や変数を / に比較・代入する場合に、これらの型が異なる場合の動作は、言語によって異なる。大きく分けるとすれば、型が異なる場合はプログラムの誤りとみなすような言語と、型が異なる場合は片方の型をもう片方の型に変換 (キャスト) するような言語である。

型のキャストの際、それぞれどのように変換されるかは言語によって異なる。

2.1.4. 関数 (function)

「関数」は、数学で用いられる「関数」とほぼ同義である。より正確に記すのであれば、「0 以上の変数を引数としてとり、0 または 1 つの値を返す」動作をするものが関数である。

関数の書き方の例を示す (書き方は言語によって異なる):

```
int example(int a, int b){
      int c = a + 2b;
return c;
}
```

この例では、2 つの整数 a と b を引数に取り、a+2b なる整数 c を返す関数を示している。

2.1.4. クラス (class)

「クラス」は、変数や関数などを含む【※】である。

2.2. 式 (expression)

式は、値を持つ命令である。

2.2.1. リテラル・変数

リテラル・変数などの値は、単体で式となる (が、言語によっては値のみの式を認めないものもある)。

式の値は、そのリテラル・変数の値である。

2.2.2. 関数の呼び出し

関数の呼び出し (関数を実行する) は、" 関数名 (引数)" という形で行うのが一般的である。

```
funcname(0, 1);
```

関数呼び出しの式は、関数の戻り値が式の値になる (戻り値がない場合は、言語により null などが値となる)。

2.2.3. 演算

演算子を用いて、演算を行うことができる。演算子によって異なるが、1 つ以上の式と演算子によって演算を行う。

2.2.3.1. 単項演算

単項演算は、式を 1 つ取る演算である。

・ 否定演算子

```
!expr //expr が真のとき式の値は偽、expr が偽のとき式の値は真
```

なお否定演算子は論理演算子の 1 つである。

・ インクリメント演算子

インクリメント演算子は変数のみを取る。

```
expr++; //expr の値を 1 増やす。式の値は変更前の expr1
++expr; //expr の値を 1 増やす。式の値は変更後の expr1
```

デクリメント演算子も同様である。

2.2.3.2. 二項演算

二項演算は、2 つの値による演算を行う。値は式で表現する。式の値は演算の結果である。二項演算を一般的に書くと次のようになる。

```
式 演算子 式
```

・ 算術演算

算術演算は、加算・乗算などの算術計算を行う。

```
4 + 5; // 式の値は 9
4 * 3; // 式の値は 12
4 * 3 + 1; // 式の値は 13
```

なおこのように副作用のない式 (変数の値が変化しないなど) は書けない言語もある。

・ 比較演算

比較演算は 2 つの値を比較する。式の値は真偽値となる。

以下は例であり、演算子は言語によって異なる。

```
expr1 == expr2 //expr1 と expr2 が等しいとき真、異なるとき偽
expr1 != expr2 //expr1 と expr2 が異なるとき真、等しいとき真
expr1 > expr2 //expr1 が expr2 より大きいとき真、そうでないとき偽
```

この他、上記に示した以外の大小の比較や厳密な比較を行う演算子が存在する。

・ 論理演算

論理演算子で一般的に用いられるのは 3 つあるが、このうち 1 つは前述の否定演算子である。残りの 2 つを示す :

```
expr1 && expr2 //expr1 と expr2 がともに真ならば真、そうでないならば偽
expr1 || expr2 //expr1 と expr2 がともに偽ならば偽、そうでないならば真
```

なお、&& 演算子は言語によっては「短絡評価」を行うことがある。この場合の定義は、expr1 が真なら値は expr2、expr1 が偽なら値は偽、となる (expr2 が関数などであるときに動作が異なる)。|| 演算子も同様である。

・ ビット演算子

省略する。

2.2.3.3. 三項演算

三項演算子は、3 つの式を取る演算子である。一般に用いられるのは条件演算子のみであるため、条件演算子を三項演算子と呼ぶことがある。

```
expr1 ? expr2 : expr3 //expr1 が真のとき expr2、expr1 が偽のとき expr3
```

2.2.3.4. 代入演算

代入演算は変数の値を変更する演算である。一般に書くと次のようになる。

```
変数 演算子 式
```

いくつか例を示す :

```
expr1 = expr2 //expr1 の値を expr2 とする。式の値は変更後の expr1
expr1 += expr2 //expr1 = expr1 + expr2 と同じ動作
```

2.2.3.5. 演算子の優先順位 (結合順位)

演算子を複数用いた場合、演算の仕方 (順序) が一意に定まらないことがある。そのような場合は、あらかじめ定められた (言語によって異なる) 演算子の優先順位によって演算を行う。

2.3. 文 (ステートメント statement)

ステートメントは、プログラムを構成する命令の単位である。ステートメント 1 つで命令を 1 つ構成すると考えても差し支えない。

プログラムのソースコードは、多数のステートメントが連なったものである。ステートメントの区切りは言語によって異なるが、多くはセミコロン ";" または改行である。

2.2.1. ブロック

多くの言語では、波括弧 "{", "}" で複数の文を囲ったものを「ブロック」という。ブロックは文と同様に扱うことができる。文 1 つを記述すべき箇所にブロックを記述することができるということである。

2.2.2. 宣言文

宣言文は、プログラムで使う変数などなどを定義・宣言する文である。

宣言文の形式は、言語によって異なる。いくつかの言語の変数定義の例を挙げる。

- C 言語

```
int i;
```

- Visual Basic

```
Dim i As Integer
```

- Javascript

```
var i;
```

2.2.3. 実行文

実行文は、プログラムが実際に実行するべき内容を記述した文である。

2.2.3.1. 式

式は文となることができる。

2.2.3.2. 制御文 (制御構造)

if 文、while 文などの文である。

これらの文は、命令の実行順序を制御するために用いられる。