ようこそCOBOLへ！

2018/08/17

伊東　輝

COBOLとは？

- 1959年に事務処理用に開発された手続き型言語であり、ソースコードの記述内容を上から順番に実行する言語である。
- 約60年前から存在する言語でありながら、未だに基本情報処理技術者の午後試験に出題され、金融系システム等のレガシーシステムでは現在もCOBOLのプログラムが稼働している。
- 今回は、COBOLのコーディングの基礎を発表する。視聴者の方々がリファレンスを引いて自力で調べられるようになることを目的とする。

漢は黙ってHELLO WORLD!

```
000010 IDENTIFICATION                    DIVISION.
000020 PROGRAM-ID.                       SAMPLE-01.
000030 AUTHOR.                           A-ITO.
000040*
000050 ENVIRONMENT                       DIVISION.
000060*
000070 DATA                              DIVISION.
000080*
000080*COMMENT!
000090 PROCEDURE                         DIVISION.
000100 MAIN.
000110     DISPLAY "HELLO WORLD!".                                    COMMENT!
000120     STOP RUN.
```

実行結果：SYSOUT(コンソール)に「HELLO WORLD!」が出力される。

DIVISIONとは？

COBOLのプログラムは、下記の4つのDIVISIONに分けて記述する。

- IDENTIFICATION　DIVISION
  プログラム名や作者等を記述する。
- ENVIRONMENT　　DIVISION
  入出力ファイルの情報等を記述する。
- DATA　　DIVISION
  ワークエリア（変数）の定義を記述する。
- PROCEDURE　　DIVISION
  ロジックを記述する。

各カラムの記述内容

COBOLのソースコードは80カラム(列)で構成され、各カラムで意味が異なる。

・1～6カラム目(一連番号領域)
　プログラム行を識別するための番号を記述する。
・7カラム目(標識領域)
　通常は空白。*をつけるとその行はコメントになる。
・8～11カラム目(A領域)
　前ページで説明した～DIVISIONや、ワークエリアのレベル(後述)、ラベル(後述)等はここから書き出す。
・12～72カラム目(B領域)
　ロジックはここに記述する。
・73～80カラム目(プログラム識別領域)
　この領域に記述した内容はコメントとして扱われる。

HELLO WORLD! で使用した命令

HELLO WORLD!では下記の命令を使用した。

・ラベル
000100 MAIN.
これはラベルを示す。ラベル自体に意味はないが、GOTO文等で制御の飛ばし先を指定する際に使用する。
・DISPLAY
オペランド(変数や定数のこと)で指定した文字列をSYSOUT(ホスト上のコンソール)へ出力する。
・STOP RUN
プログラムを終了させる。CやjavaでいうとExit(0)。

ピリオドの機能

ピリオドは文の終わりを示す機能がある。
例えばIF文の場合は、下記のように最後のEND-IFにピリオドを打つ（途中でピリオドを打つとコンパイルエラー）。

```
000010      IF  FLAG = 1 THEN
000020          DISPLAY  "HELLO WORLD!"
000030      END-IF.
```

下記のように最後のEND-IFを省略してピリオドで止めても処理内容としては同じになる。
ただし、ピリオドを打ち忘れると、以降の処理はIF文の中として扱われてしまう（バグを生み出す要因となる）。

```
000010      IF  FLAG = 1 THEN
000020          DISPLAY  "HELLO WORLD!".
```

ファイル出力プログラムの例(1/3)

```
000010 IDENTIFICATION                          DIVISION.
000020 PROGRAM-ID.                             SAMPLE-02.
000030 AUTHOR.                                 A-ITO.
000040*
000050 ENVIRONMENT                             DIVISION.
000060 INPUT-OUTPUT                            SECTION.
000070 FILE-CONTROL.
000080 SELECT F1 ASSIGN TO ASSIGNMENT-NAME.
000090*
000100  DATA                                   DIVISION.
000110  FILE                                   SECTION.
000120  FD  F1.
000130  01  F1R.
000140      03  F1-HELLO      PIC X(05).
000150      03  FILLER        PIC X(01) VALUE SPACE.
000160      03  F1-WORLD      PIC X(06).
000170  WORKING-STORAGE                        SECTION.
000180  01  WK-CONSTANT.
000190      03  WK-CON-HELLO  PIC X(05) VALUE "HELLO".
000200      03  WK-CON-WORLD  PIC X(06) VALUE "WORLD!".
```

ファイル出力プログラムの例(2/3)

```
000210*
000220  PROCEDURE                              DIVISION.
000230*
000240* 各種処理の呼び出し
000250*
000260 000-CONTROLLER-S.
000270     PERFORM  100-START-S THRU 100-START-E.
000280     PERFORM  200-MAIN-S  THRU 200-MAIN-E.
000290     PERFORM  300-END-S   THRU 300-END-E.
000300     STOP RUN.
000310 000-CONTROLLER-E.
000320*
000330* 前処理
000340*
000350 100-START-S.
000360     INITIALIZE F1R.
000370     OPEN  OUTPUT  F1.
000380 100-START-E.
```

ファイル出力プログラムの例(3/3)

```
000390*
000400* 主処理
000410*
000420 200-MAIN-S.
000430     MOVE WK-CON-HELLO TO F1-HELLO.
000440     MOVE WK-CON-WORLD TO F1-WORLD.
000450     WRITE F1 FROM F1R.
000460 200-MAIN-E.
000470*
000480* 後処理
000490*
000500 300-END-S.
000510     CLOSE  F1.
000520 300-END-E.
```

実行結果：ASSIGNMENT-NAMEで示したファイルに下記レコードが出力される。
HELLO WORLD!

ENVIRONMENT DIVISION の記述内容

今回、ENVIRONENT DIVISIONにて出力ファイルの定義を行った。

```
000060 INPUT-OUTPUT                                    SECTION.
000070 FILE-CONTROL.
000080 SELECT F1 ASSIGN TO ASSIGNMENT-NAME.
```

「INPUT-OUTPUT SECTION」と「FILE-CONTROL」は入出力ファイルの定義を行う場合の見出しである。
今回の例で言う「F1」は出力ファイル名を示し、ソースコード中ではこの名前を使用する。
ASSIGNMENT-NAMEには、ファイルの物理的な場所を示すパラメータを与える。どのようなOSを使用しているかによってパラメータの与え方は異なり、ホストであればJCL(ジョブ制御言語)で定義したパラメータを与える。

DATA DIVISION の見出しとFDの説明

DATA DIVISIONではワークエリアの定義を行った。

「FILE SECTION」はファイル入出力に関するワークエリアの定義を行う場合の見出しであり、「WORKING-STORAGE SECTION」は計算や文字列操作等を行う際に利用するワークエリアの定義を行う場合の見出しである。

000120　FD　F1.
「FD」は入出力ファイルを読み書きするためのワークエリアであり、今回の例ではこのワークエリアに書き込みを行うことでF1ファイルへの出力が行われる。

ワークエリアのレベル

```
000130   01   F1R.
000140        03   F1-HELLO        PIC X(05).
000150        03   FILLER          PIC X(01) VALUE SPACE.
000160        03   F1-WORLD        PIC X(06).
```

「01」や「03」はワークエリアのレベルを指す。下位レベル(03)のワークエリアは上位レベル(01)の下位項目となる。ロジック中で上位レベルのワークエリアを参照すると、下位項目のワークエリアが連結されたワークエリアとして扱われる。
なお、「FILLER」は特別なワークエリア名であり、ソースコード中で何度でも定義できるが、ロジック中での参照はできない。

ワークエリアのタイプと定数

```
000130   01   F1R.
000140        03   F1-HELLO        PIC X(05).
000150        03   FILLER          PIC X(01) VALUE SPACE.
000160        03   F1-WORLD        PIC X(06).
```

ここで、「X」は1バイトの文字タイプを指し、「X(05)」は5バイトの文字列を意味する。
今回は使用していないが、「9」は1バイトの数値タイプを指し、数値計算をする際は9タイプを指定する必要がある。
他にも、ゼロサプレスを自動的に行うタイプや、符号を自動的に付与してくれるタイプ等が存在する。

なお、VALUE句を使用することで定数として利用できる。

PERFORM～THRU～文による制御

PERFORM句でラベルを指定するとそのラベルに制御を飛ばすことができ、THRU句で指定したラベルに到達した時に呼び出し元に制御を戻すことができる。
他言語で言う関数の動きはこれで疑似的に実現できる。

```
000270       PERFORM  100-START-S THRU 100-START-E.
```

この例では、100-START-Sのラベルの箇所へ処理を飛ばした後、100-START-Eのラベルの箇所まで到達した際に呼び出し元へ戻る。

なお、一般的にCOBOLには構造化分析のデザイン手法が用いられるため、PERFORM～THRU～文で「前処理」「主処理」「後処理」に3分割することが多い。

前処理の内容の説明

前処理ではワークエリア初期化やファイル等のオープンを行う。

```
000360       INITIALIZE F1R.
000370       OPEN  OUTPUT  F1.
```

INITIALIZE文ではワークエリアの初期化を行う。Xタイプのワークエリアには SPACE、9タイプのワークエリアには0がセットされるが、FILLERには初期化が適用されない。

OPEN文はファイルオープンを行う。
入力ファイルに関してはINPUT句、出力ファイルに関してはOUTPUT句を用いる。

主処理の内容の説明

主処理には、ファイル読み込みやファイル書き込み、及びビジネスロジックの記述を行う。

```
000430       MOVE WK-CON-HELLO TO F1-HELLO.
000440       MOVE WK-CON-WORLD TO F1-WORLD.
000450       WRITE F1 FROM F1R.
```

MOVE文はワークエリアの代入を行う。000430の行では、WK-CON-HELLOの内容(“HELLO”)をF1-HELLOへ代入している。

WRITE文はファイルへの書き込みを行う。書き込む内容はFROM句で指定する。今回の例ではF1Rのワークエリアの内容をF1ファイルへ書き込んでいる。

後処理の内容の説明

後処理ではファイル等のクローズを行う。

```
000510        CLOSE   F1.
```

CLOSE文はファイルクローズを行う。

なお、今回の例では記述していないが、処理件数をSYSOUT等へ出力する際は、後処理で行うことが多い。

ワークエリアは全てグローバル！

COBOLのワークエリアは全てグローバル変数であり、変数のスコープに相当する概念が存在しないことに注意が必要。

COBOLでは、COPY文を使用して複数のソースファイルを結合することができる。
一般的に、複数のプログラムで使用する共通処理をソースファイルに外出しし、COPY文で結合する。
しかし、共通処理内で定義しているワークエリアは呼び出し元のソースファイルから参照することができ、その逆もできるため、影響範囲を限定することができない。
上記の仕様があるため、COBOLで保守性を確保するためには、コーディングルールの整備と遵守が必要である。

筆者が考えるCOBOLの利点

COBOLの一番の利点は、固定長のファイルを入出力することが容易である点にあると思う。
特に、帳票の出力はCOBOLが圧倒的に楽だと感じる。
レコードの階層関係やタイプやバイト数が、ワークエリアの定義を見れば一目瞭然のため、可読性が高い。
また、ワークエリアの上位項目と下位項目を上手く定義できればMOVE文だけで文字列の結合・分割が可能であるし、ゼロサプレスや符号の付与等の編集も処理を作りこむことなくワークエリアの定義だけで行うことができる。

なお、他のサイトでは、小数点以下の数値計算が他の言語よりも容易であるという意見が多い。
元々事務処理用に開発された言語であるため、現在でも事務処理においては一定の合理性がある言語だと思う。

ご清聴ありがとうございました。