１．概要

プログラム開発手法は様々なものがあるがここでは、代表的なフローチャートとＰＡＤを記述するものとする。また、メルマガで取られていたＰＩＣ図を参考に掲げるものとする。

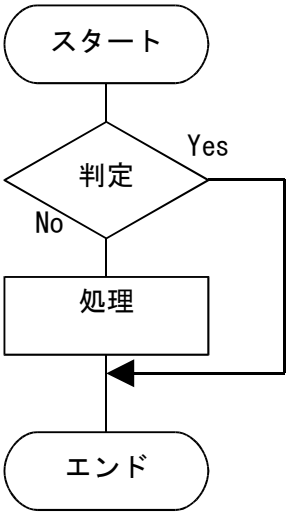
２．一般的な開発工程（作業内容とその内訳）

見積項目	項番	作業内容	内　訳
基本設計	１	基本設計	1 参考資料調査・検討 2 基本設計及び設計書作成
詳細設計	２	詳細設計	1 基本設計書検討 2 詳細設計及び設計書作成 3 フローチャート・ＰＡＤ作成
コーディング	３	コーディング	1 コーディング 2 チェックリスト作成（ＵＤ用、ＣＤ用）
デバッグ＆テスト	４	アセンブル／コンパイル	1 ソース・プログラム作成 2 アセンブル／コンパイル 3 ロード・モジュール作成（リンク）
	５	デバッグ＆テスト	1 デバッグ機器手配 2 デバッグ操作習得 3 机上デバッグ 4 マシンデバッグ（ＵＤ、ＣＤ）
	６	システムテスト	1 チェック・リスト作成（ＳＤ用） 2 システム・テスト（機能テスト） 3 立会い検査 4 現地テスト
その他	７	ドキュメント整理	①　設計書の訂正・修正
	８	説明書作成	1 承認仕様書作成 2 機能仕様書作成 3 操作説明書作成 4 取扱説明書作成
	９	会議	全ての会議、打合せ
	１０	出張往復時間	項番１～９までに必要な出張時間

ＵＤ：単体チェック、ＣＤ：結合チェック、ＳＤ：システムチェック

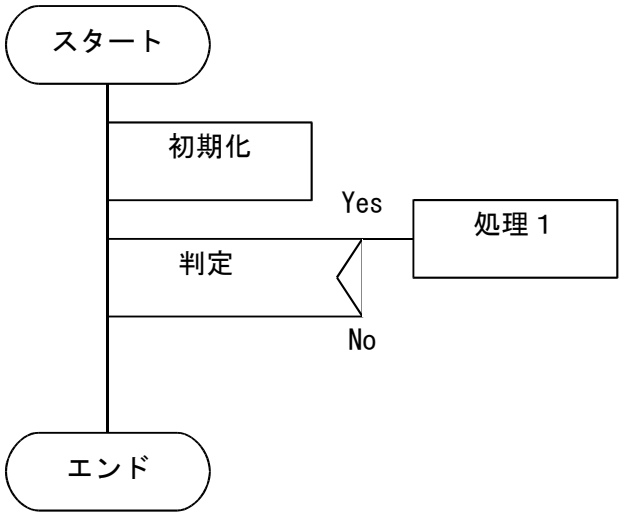
以上のような手順を経てソフトウェアが作成されるのですが、この中の詳細設計についての手法を以下に示します。なお、フローチャートとＰＡＤについては既存の知識として簡単に述べてあります。
ここでは、ぷるさり氏が提唱する手法を主眼にご紹介してあることをお断りしておきます。

３．フローチャート



図からわかるように、論理の組み方を立体的に記述する方法です。
但し、問題点としては分岐処理がｇｏｔｏ文になるためリストを見るときは煩雑になる傾向があります。

４．ＰＡＤ（構造化プログラミング）



構造化プログラミングの良いところは上から下へ処理が流れ、分岐が飛ばない（基本的に）ところがメンテナンスしやすさにつながっています。
但し、横に広がるので５０ステップモジュールとか１００ステップモジュールを意識しないとドキュメントも見難く、リストも見難くなります。

５．ＰＩＣ図

――――――――――――――――――――――――――――――――――［ＰＩＣ図］

ＰＩＣ図とはぷるさりがこのメルマガ用に考えた、プログラムの構造化表記技法のことで、
“Purusaly's structured indication chart” の略です。
DATA DIVISION. の　PIC とは全く関係ありませんので、あらかじめ。

なぜこのような物を考えたかと申しますと、

・メルマガでフローチャートは表現しづらい
・フローチャートは構造化の表記には適していない

などがあります。

あと、構造化の表記としては“ＰＡＤ”図が比較的有名ですが、これはどうしても表記が「横」に広がって行きやすく、また、記号を書く段階で、その階層がどのくらいの大きさや深さになるか、考えながら書かなくてはならないこと。そして「階層が深くなった場合、図からコードを起こす時、ミスを起こしやすい」ということから、このメルマガでは採用しませんでした。

ということで、このメルマガでは、処理手順を表す際、ぷるさりオリジナルのＰＩＣ図にて表記しますので、皆さんも覚えてネ！

━━…………・・・・・・

■ＰＩＣ図の書き方

・・・・・………━━

ＰＩＣ図は構造化理論に基づいて作られていますので、基本は
・順次
・選択
・繰り返し

の３つだけです。

これをパソコンの罫線や記号を用いて表わしたものです

これにＣＯＢＯＬ８５独自の拡張機能をいくつか付け足しただけです。

では、気楽にいってみましょう！

……

●1.順次(SEQUENCE)　　＊処理は上から下へ

（例）

```
│                        │
├ 処理Ａ                 ├ 在庫数＝現在庫数－本日販売個数
├ 処理Ｂ                 ├ 売上金額＝売上金額+(製品単価＊本日販売個数)
│                        │
```

……

●2.選択(IFTHENELSE,CASE)

選択は制御記号に「△」を使い、その右に　IFTHENELSE　の場合は「IF:」と書き、CASE　の場合は「CASE:」と書き、さらにその右に、判断／選択条件を書きます。条件の真偽は「○－」マークを書き、その右に YES/NO　または　真の場合の条件を書きます

・IFTHENELSE

```
|
△IF: 入庫データか？
|〇-YES
| ├  在庫数＝在庫数＋個数
|〇 -NO
| └  在庫数＝在庫数－個数
```

・ IFTHENELSE　の入れ子

```
|
△ IF: XA = "A"　か？
|〇-YES
| ├△ IF: A > MAX　か？
| |〇-YES
| | ├   B = MAX
| |〇 -NO
| | └   B = A * 0.8
| ├   U = A - B
|〇 -NO
| └   U = A
├   USUM = USUM  + U
|
```

・CASE

```
|
△CASE: 割引区分
|〇-1
| ├  割引区分が１の時の処理
|〇-2
| ├  割引区分が２の時の処理
|〇-3
| ├  割引区分が３の時の処理
|〇-4
| └  割引区分が４の時の処理
|
```

ＣＯＢＯＬの場合、 CASE　は　 EVALUATE　になります。
（この場合〇- の　- が、〇- の記号なのか、それとも「マイナス」なのか、あいまいになるので、改良の余地がありますね…）

このように、処理などの内容は文章でもよければ、「　A > MAX 」のように表現してもよいものとします。要は分かりやすければいいのです。

……

●3.繰り返し(DOWHILE,DOUNTIL)

繰り返しは制御記号に「◎」を使い、その右に　DOWHILE の場合は「W:」と書き、DOUNTIL の場合は「U:」と書き、さらにその右に繰り返し条件を書きます。

<<解説 >>---------- ここから ----------<< 解説>>

*** DOWHILE とは　***

条件が真である間、処理を繰り返すのが　DOWHILE です。
また、条件（ループ脱出）の判断をループ処理を実行する前に行うのがDOWHILE の特徴です。
ＣＯＢＯＬ８５で　DOWHILE 型の処理を実行する場合は　PERFORM を命令を使用します。
PERFORM 命令は特に指示がなければ　PERFORM 命令開始時にループ判断を行いますので、そのままでも良いのですが、特にそのことを意図的に明示したい場合は、 WITH TEST BEFORE　句を指定します。

/*-- MEMO --*/
ＣＯＢＯＬ８５には、直接　DOWHILE を実行する命令はないので、 PERFORMUNTIL　句で代用し、ループ脱出条件をひっくり返して指定します。
/*----------*/

*** DOUNTIL とは　***

逆に条件が真になるまで（つまり偽の間）処理を繰り返すのが　DOUNTIL です。
DOUNTIL は　DOWHILE とは逆に、条件（ループ脱出）の判断をループ処理を実行した後に行うのが特徴です。つまり、最低１回は必ず処理が行われる訳です。
ＣＯＢＯＬ８５で　DOUNTIL 型の処理を実行する場合も　DOWHILE 同様　PERFORM 命令を使用しますが、その際、 WITH TEST AFTER　句を指定します。

<<解説 >>---------- ここまで ----------<< 解説>>

・DOWHILE　　*在庫が無くなるまで注文を受け付ける

```
 ├  ZAIKO = F-ZAIKO
//
//（いろいろ処理が入って…）
 |
 ├   ACCEPT CHUMON
◎ W: ZAIKO >= CHUMON
 | ├   ZAIKO = ZAIKO - CHUMON
 | ├   DISPLAY " 注文を受け付けました"
 | └   ACCEPT CHUMON
```

・DOUNTIL　　*1 ～ 3 の入力が得られるまで（画面からの操作指示など）

```
|
◎U: ( 1 <= SYORI-CODE  ) AND ( SYORI-CODE <= 3 )
| └  ACCEPT SYORI-CODE
```

………………………………………………………………………………

基本はこれだけです。
その他にＣＯＢＯＬ特有の命令として

………………………………………………………………………………

●4. 例外処理を伴う命令

READ 命令など　AT END　が付くような命令は、制御記号に「▼」を使い、その右に処理内容を書き、例外時の処理は　IFTHENELSE　の時と同様にその下に書きます。

・例外処理

```
|
▼売り上げトランザクションを読む
| ○ -AT END
| └   SET URI-TRAN-END TO TRUE
```

もちろんその下に　○-NOT AT END　を書いてもかまいません。

―――――――――――――――――――――――――――――――――――

その他

―――――――――――――――――――――――――――――――――――

●5. 開始終了

┳　と　┻　を使用する。
場合によってはその上下に　START, END　などと書いてもよい。

売り上げ処理

```
START
 ┳
 ├
 ├
 //
 //
 ├
 ├
```

```
┻
END
```

……

●6.結合

用紙の都合などでＰＩＣ図が１列に収まらないような場合は「▽」を使いＰＩＣ図を複数列に分ける。（GO TO の意味ではありません）

```
START
┯
├
├              :A
├              ▽
//      ├
//      ├
├              ┻
├              END
▽
To:A
```

……

●7.まとまった処理

機能単位のまとまった処理は　［　］でくくって、処理内容を記入しましょう。

```
|
├[ 在庫数確認　]
├[ 発注処理　　]
|
```

……

●8.まとまった処理と処理内容を一緒に記入する

まあ、こんな感じになるでしょうか…

```
|
├[ 在庫数確認　]
|   ├　在庫数＝在庫数－本日販売個数
|   └　在庫数＝在庫数＋本日入荷数
├[ 発注処理　　]
|   △IF: 在庫数が最低確保在庫数を下回ったか？
|     ○-YES
|      ├　発注数＝最大確保数－在庫数
|     ○ -NO
```

｜　　└　発注数＝０

以上がＰＩＣ図（第１案）です。

フローチャートしか知らない方や、ＰＡＤなどに慣れ親しんでいる方にとっては、ＰＩＣ図はそれほど「ビジュアル」ではないので、慣れないうちは余計分かりづらいかも知れません。

でもぷるさりは、このような記述法がＣＯＢＯＬ８５などで「構造化プログラミング」によりソフトウェアを開発する場合、最も効率が良いと考えています。

一度、いらなくなったコピー用紙の裏側にでも、シャーペンなんかで手書きで書いてみてください。「手書き」と言うのがポイントです。ムリしてパソコンで書こうなどと思わないでください！（ぷるさりも実際にＰＩＣ図を書く時は紙にフリーハンドで書いています。）

すぐに慣れると思います。そして、フローチャートよりも、ＰＡＤよりもずっと書きやすいことを実感されることでしょう。

本当は　IF　や　CASE ,WHILE　などの命令は、その文字自体を記号で囲んでしまえば、もっと直感的に見やすく、分かりやすくなるのですが、メルマガでそのようなことに時間を費やしていては本末転倒というもの。とにかくメルマガという媒体の特性上、パソコンでカンタンに入力ができなければ意味がない、と思い、とりあえず（第１案）ではこのようなカタチにしました。

6 .