

1. ABAP/4 Introduction

📖 Introduction

ABAP/4(Advanced Business Application Programming)是 SAP/R3 目前唯一的系统发展工具, 属 4GL, 语法比较近似 Visual Basic 或 JAVA, 和传统 third-generation 语言, 如 C,PASCAL 有很大不同, 在程序模块(Program Structure Module)可分以下三个部分:

1.Sequential coding within processing block

与一般语言语法近似, 如 IF,WHILE 等, 但并没有 GOTO 叙述

2.Reports

呼叫一个独立的事件(Depending Event), 读取 database 产生数据列表

3.Dialog

屏幕参数输入的对话框, 专门处理 database 读取或异动的 transaction process

📖 Basic Language Overview

- 1.data element 宣告方式, 如数值, 字符数据变量宣告
- 2.操作数(operate)使用, 如 + - * /
- 3.Control element 使用, 如 Boolean 值
- 4.特殊数据格式, 如日期与时间
- 5.字符串字料处理 function, 如部分字符串的截取
- 6.子程序或自定函数的呼叫
- 7.SQL 语法使用
- 8.数据结构的使用, 如 process internal table 的宣告与使用

📖 Reports Overview

- 1.Reports Task, 如报表屏幕预览或打印机打印的选择
- 2.Reports 模块是一个 Stand-alone 程序,
- 3.database 读取方式, 如可定义 logical database(与磁盘的 physical storage 对映)
- 4.报表数据的计算与产生
- 5.报表的输出

📖 Dialog Overview

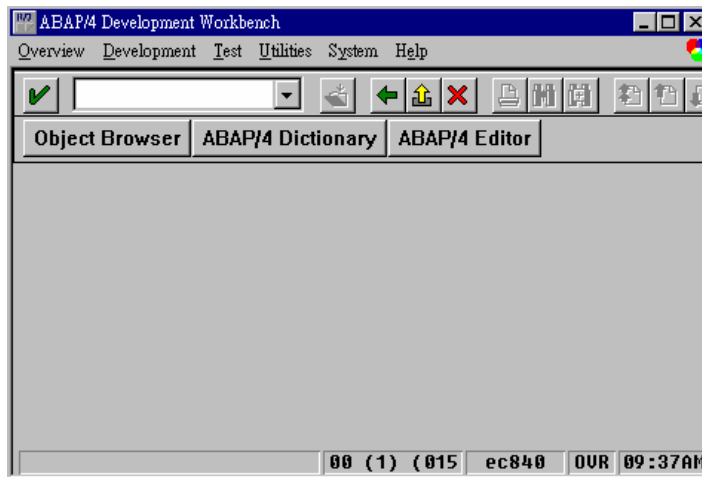
- 1.专处理 database 的读取与异动, 如使用 SQL 指令
- 2.Dialog 不是一个 Stand-Along Program, 使用 transaction code 来产生屏幕对话框
- 3.由 flow logic 控制, flow logic 分成 PBO(Process Before Output)与 PAI(Process After Input)

2.Begin To Programming

2.1 ABAP/4 Editor

📖Creating ABAP/4 Program

使用 ABAP Workbench 撰写程序(Choose Tools->ABAP/4 Workbench, Transaction Code: S001),
萤



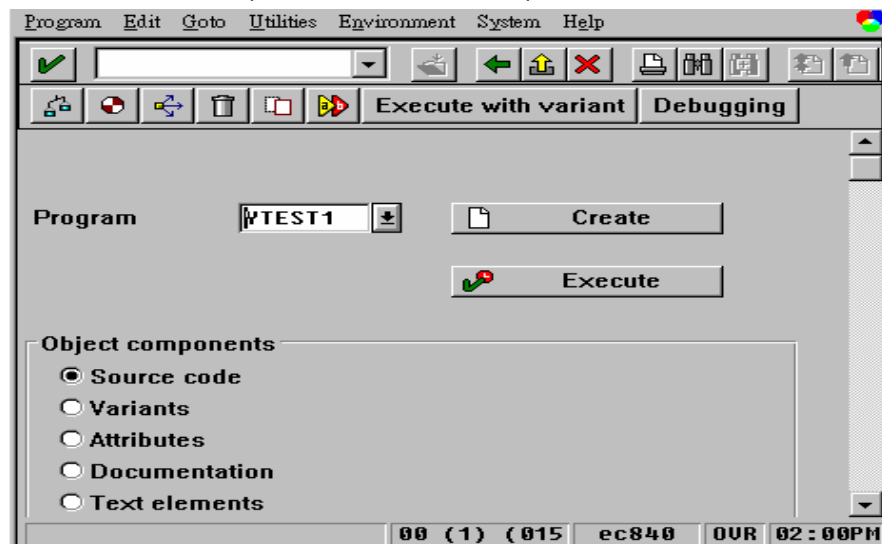
幕如下:

可分成:

- 1.ABAP/4 Editor :针对简单的报表或程序, 仅使用几个组件或不使用
- 2.Object Browser :针对复杂的报表或程序, 如 Dialog Transaction Module 撰写

📖使用 ABAP/4 Editor 撰写程序

1.ABAP/4 Editor 画面如下(Transaction Code:SE38):



2.输入程序名称, 如果是新程序, 按下"Create", 如果修改已存在程序, 则按下"Change"或 F6 键
在命名规则上, Reports 程序为 Yaxxxxxx 或 Zaxxxxxx, a 表 application module 简称, 如 s 表 SD
Dialog 程序为 SAPMYxxx 或 SAPMZxxx

3.输入程序 Attribute

Title	
Maintenance language	E English

Attributes	
Type	?
Status	
Application	?
Authorization group	
Development class	

(1).Title:程序描述或功能说明

(2).Type:Execute mode: 1: Stand-alone Program 如 Reports

I: Include Program

M: Module Pool

F: Function Group

S: Subroutine Pool

(3).Status:Program development status: P: SAP standard production program

K: Customer production program

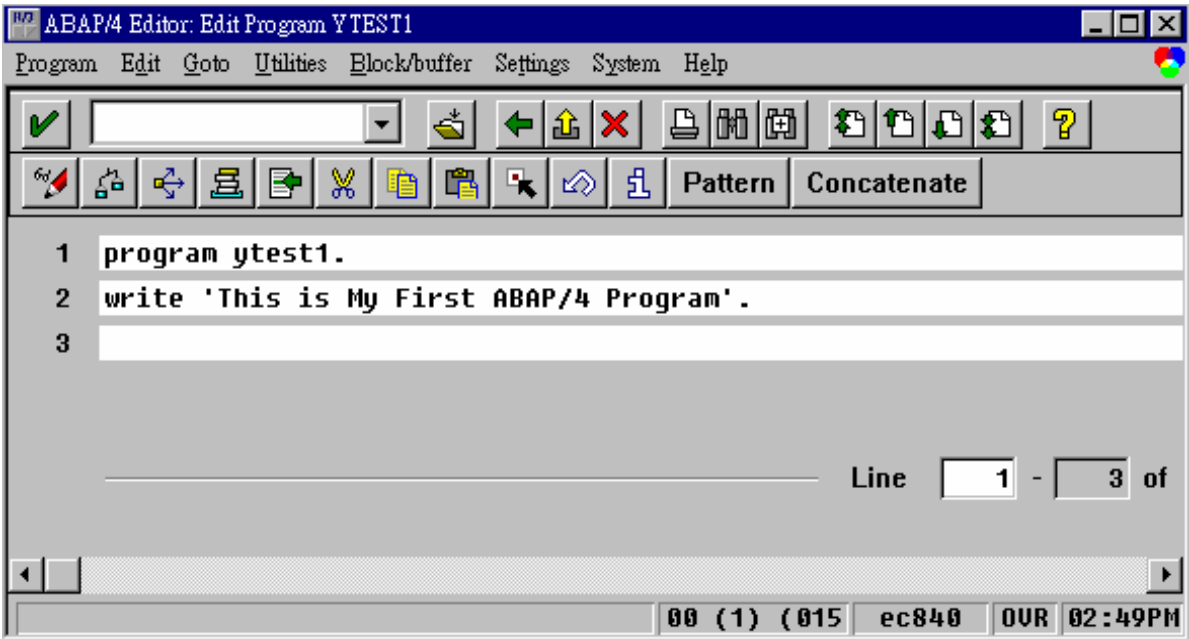
S: System program

T: Test program

(4).Application: 程序所属的 application module, 如 F 表 Financial accounting

(5).Development class: 用于同一系统中各个 program, 如果不属任一 class, 可使用\$TMP

4.撰写 Source Code



Program 之后接的是程序名称, Write 是显示的意思, 会将所接的字符串在屏幕上显示, 注意每
—
行最后要有一个 . (点), 表示叙述的结束, 储存后回 ABAP/4 Editor 画面

5. 执行程序

按"Execute"或 F8 键, 屏幕可见执行所得的结果

📖重要的编辑键

key	Function
F8	执行程序
F5	复制游标所在列的内容
F11	储存档案
CTRL+F11	删除游标所在列

2.2 ABAP/4 Data Element

Data Type

ABAP/4 的数据型态可分成:

Type	Length	Range	Initial Value	Description
C	1	1-65535Byte	Space	字符串数据, 如'Program'
D	8	8Byte	'00000000'	日期数据, 格式为 YYYYMMDD
F	8	8Byte	0	浮点数
I	4	-2 ³¹ 至 2 ³¹ -1	0	整数
N	1	1-65535Byte	'00...0'	数值所组成的字符串
P	8	1-16Byte	0	Packed 数, 用在小数点数
T	6	6Byte	'000000'	时间数据, 格式为 HHMMSS
X	1	1-65535Byte	X'00'	16 进位数

变量宣告

变量宣告包含 name, type, length 和 structure 四个部分, 使用 DATA 指令, 如

```
DATA: S1 TYPE I,  
      SUM TYPE I.
```

常数宣告

常数宣告使用 CONSTANTS 指令, 如宣告 PI 是一个小数点 5 位的值 3.14159,

```
CONSTANTS PI TYPE P DECIMALS 5 VALUE '3.14159'.
```

系统所定义数据

这是由系统所定义的专有名词, 如

```
SPACE      空白字符串  
SY-SUBRC   系统执行传回值, 0 表示成功  
SY-UNAME   logon 账号  
SY-DATUM   系统日期  
SY-UZEIT   系统时间  
SY-TCODE   目前的 transaction code
```

▣TYPE 叙述

用来指定数据类型或宣告自定义数据类型

Example:

```
TYPES: BEGIN OF MYLIST,
        NAME(10) TYPE C,
        NUMBER TYPE I,
        END OF MYLIST.
DATA LIST TYPE MYLIST.
```

▣LIKE 叙述

跟 TYPE 叙述使用格式相同, 如

```
DATA TRANSCODE LIKE SY-TCODE.
```

不同的是 LIKE 用在已有值的数据项, 如系统变量, 而 TYPE 叙述则是用在指定数据类型。

▣DATA 叙述

语法:

```
DATA <f> [<length>] <type> [<value>] [<decimals>]
```

<f>: 变量名称, 最长 30 个字符, 不可含有 + . , : () 等字符

<length><type>: 数据类型及长度, 如 LINE(20) TYPE C. MYNAME LIKE SY-UNAME.

<value>: 初值

<decimals>: 小数位数

Example:

```
DATA: COUNTER TYPE P VALUE 1,
      FLAG TYPE C VALUE IS INITIAL,
      WEIGHT TYPE P DECIMALS 2 VALUE '1.25'.
```

字段变量的宣告:

```
DATA: BEGIN OF ADDRESS,
      NAME(10) TYPE C,
      NUMBER TYPE P,
      END OF ADDRESS.
```

使用时用字段变量加上组件名称, 如 ADDRESS-NAME

CONSTANTS 叙述

用来宣告常数

语法:

```
CONSTANTS <c> [<length>] <type> [<value>] [<decimals>]
```

Example:

```
CONSTANTS: CNAME(10)  VALUE '周庆日',  
           BIRTH_DAY  TYPE  D  VALUE '19650201'.
```

STATICS 叙述

宣告的变量仅在目前的程序中使用, 结束后会自动释放

语法:

```
STATICS <c> [<length>] <type> [<value>] [<decimals>]
```

TABLES 叙述

用来宣告 Table Work Area 的数据, 对映至 ABAP/4 资料文件(Dictionary Object),
由 SQL 指令加载所需数据

语法:

```
TABLES <dbtab>
```

Example:

```
TABLES: SPFL.  
SELECT * FROM SPFL.  
       WRITE: SPFL-MANDT, SPFL-CARRID, SPFL-CONNECTION.  
ENDSELECT.
```

从 ABAP/4 Dictionary 的 SPFL 档载入 MANDT, CARRID, CONNECTION 三个字段至
SPFL 此 Table Work Area

2.3 Outputting Data to Screen

Write 叙述

ABAP/4 用来在屏幕上输出数据的指令是 Write 指令

语法:

Write 资料项

数据项可以是常数或变量, 如:

WRITE 'This is sample'.

WRITE: 'COMPANY:',STFL-CARRID.

指定屏幕位置显示

语法:

Write AT [/] [<pos>] [<len>] 资料项

/: 先往下一列

pos: 屏幕 X 轴坐标

(len): 显示数据的长度

Example:

WRITE 'First Line '.

WRITE / 6 'Second Line'.

输出结果:

First Line

Second Line

DATA: NUMBER TYPE I VALUE '1234567890'.

TEXT(10) VALUE 'ABCDEFGHJI'.

WRITE: (5) NUMBER, /(6) TEXT.

输出结果:

*7890

ABCDEF

指定显示格式

语法:

WRITE 数据项 <显示格式参数>

显示格式参数:

LEFT-JUSTIFIED

数据靠左显示

CENTERED	数据靠中间显示
RIGHT-JUSTIFIED	数据靠右显示
UNDER <g>	在数据项<g>的 X 轴开始坐标显示
NO-GAP	紧接着显示, 不留空格
USING EDIT MASK <m>	使用内嵌字符显示, 如 11:20:30
USING NO EDIT MASK	不使用内嵌字符
NO-ZERO	数字前面 0 的部分不显示
NO-SIGN	不显示正负号
DECIMALS <d>	显示 d 位小数字数
EXPONENT <e>	F(浮点数) exponent 的值
ROUND <r>	四舍五入至小数位数下 r 位
CURRENCY <c>	币别显示
DD/MM/YY	日期显示格式
MM/DD/YY	
DD/MM/YYYY	
MM/DD/YYYY	
DDMMYY	
MMDDYY	
YYMMDD	

Example:

```
DATA: X TYPE I VALUE '112030',
      A(5) VALUE 'ABCDE'.
WRITE X USING EDIT MASK '__:__:__'.
输出结果为 11:20:30

WRITE X USING EDIT MASK '$____,____'
输出结果为 $112,030
```

产生空白列

产生 n 个空白列

语法:

```
SKIP [<n>]
```

Example:

```
WRITE 'PASS1'.
SKIP.
WRITE 'PASS2'.
```

输出结果为:

PASS1

PASS2

显示图标

可以显示 R/3 系统所提供的符号或图标

语法:

```
WRITE <symbol-name> AS SYMBOL
```

```
WRITE <icon-name> AS ICON
```

Example:

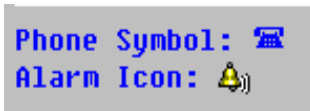
```
INCLUDE <SYMBOL>
```

```
INCLUDE <ICON>
```

```
WRITE: / 'Phone Symbol:', SYM_PHONE AS SYMBOL.
```

```
WRITE: / 'Alarm Icon:', ICON_ALARM AS ICON.
```

执行结果:



```
Phone Symbol: ☎  
Alarm Icon: 🔔
```

要查看系统所提供有那些符号及图标, 可选择 “Edit” 下的 “Insert Statement”, 选择 “WRITE”

接下来选择要查的群组, 如 SYMBOL 或 ICON, 按下 “Display” 即可.

跳至指定列坐标

将坐标跳至指定的 Y 轴列坐标

语法:

```
SKIP TO LINE [<n>]
```

Example:

```
SKIP TO LINE 5.
```

```
WRITE 'PASS1'.
```

显示 CHECK BOX 数据

以字符串数据内容的第一个字符为 CHECK BOX 的输出, 如果是空白, CHECK BOX 显示为空白, 相反则显示 X, 可用在逻辑判断检查
语法:

```
WRITE <资料项> AS CHECKBOX.
```

Example:

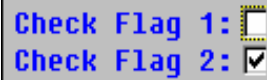
```
DATA: FLAG1 VALUE ' '.
```

```
      FLAG2 VALUE 'X'.
```

```
WRITE: / 'CHECK FLAG 1:', FLAG1 AS CHECKBOX.
```

```
WRITE: / 'CHECK FLAG 2:', FLAG2 AS CHECKBOX.
```

执行结果:



```
Check Flag 1: ☐  
Check Flag 2: ☒
```

2.4 Processing Data

Assign Value

语法:

```
MOVE <F1> TO <F2>
```

将 F1 的值存至变数 F2 中, 也可写成 $F2 = F1$

Example:

```
M_NAME = 'CHER'.
```

使用 Offset

语法:

```
MOVE <F1>[+<O1>] TO <F2>[+<O2>]
```

Example:

```
DATA: F1(10) VALUE 'ABCDEFGHIJ'.
```

```
F2(5).
```

F2 = F1+3(5). “自第 4 个位置开始取出 5 个字符

F2 的内容会变成 DEFGH

Field String 组件的复制

语法:

```
MOVE -CORRESPONDING <Strings1> TO <String2>.
```

将 Strings1 中的 field 组件的数据复制至 String2 中, 仅复制相同名称的组件

Example:

```
DATA: BEGIN OF ADDRESS,  
      FIRSTNAME(10) VALUE 'LULU',  
      LASTNAME(10) VALUE 'CHOU',  
      TEL(12) VALUE '4660570',  
      END OF ADDRESS.
```

```
DATA: BEGIN OF NAME,  
      FIRSTNAME(10),  
      LASTNAME(10),  
      E_MAIL(30),  
      END OF NAME.
```

```
MOVE-CORRESPONDING ADDRESS TO NAME.
```

NAME-FIRSTNAME 变成 'LULU', NAME-LASTNAME 变成 'CHOU',

而 NAME-E_MAIL 则不变

☐变量 CALL BY VALUE 的使用

在变量的使用上, 可以使用类似 Call By Value 的方法

语法:

```
WRITE (<f>) TO <g>
```

Example:

```
DATA: NAME(20) VALUE 'SOURCE',  
      SOURCE(10) VALUE 'LILY',  
      TARGET(10).
```

```
WRITE (NAME) TO TARGET.
```

```
WRITE / TARGET.
```

屏幕可印出 LILY

☐清除变量内容

语法:

```
CLEAR <f>
```

清除变量现在内容, 恢复成初值

Example:

```
DATA N TYPE I VALUE 100.
```

```
CLEAR N.
```

变量 N 的内容变成 0

☐算术符号

** 乘幂

* 乘

/ 除

+

加

- 减

DIV 整数除法

MOD 余数除法

☐数值函数

1. ABS(N): 传回数值 N 的绝对值

2. SIGN(N): 1 if $N > 0$

0 if $N = 0$

-1 if $N < 0$

3. CEIL(N): 传回大于数值 N 的最小整数

Example:

WRITE CEIL(-5.65) 印出 -5.00

WRITE CELL(4.54) 印出 5.00

4. FLOOR(N): 传回小于数值 N 的最大整数

Example:

WRITE FLOOR(-5.65) 印出 -6.00

WRITE FLOOR(4.54) 印出 4.00

5. TRUNC(N): 传回数值 N 的整数部分

Example:

WRITE TRUNC(5.65) 印出 5.00

6. FRAC(N): 传回数值 N 的小数部分

Example:

WRITE FRAC(5.65) 印出 0.65

7. COS(A), SIN(A), TAN(A): 传回三角函数 $\cos A$, $\sin A$, $\tan A$ 的值, A 为弧度量

8. EXP(N): 传回 e^N 值

8. LOG(N): 传回 $\log eN$ 值

9. LOG10(N): 传回 $\log N$ 值

10. SQRT(N): 传回 N 的平方根值

☐日期与时间运算

1. 日期数据的运算

日期数据可以直接运算, 如加法与减法的运算

Example:

DATA: Mdate TYPE D.

Mdate = SY-DATUM. " 如传回 19971015

Mdate+6(2) = '01' " Mdate 变成 19971001

Mdate = Mdate - 1 " Mdate 变成 19970931

2.时间数据的运算

时间格式为 'hhmmss', 如 '212030' 表 '21:20:30'

Example:

```
DATA: HOURS  TYPE  I,  
      MINUTES TYPE  I,  
      T2  TYPE  T  VALUE '200000',  
      T1  TYPE  T  VALUE '183000'.  
HOURS = (T2 - T1) / 3600. "计算有几小时  
MINUTES = (T2 - T1) / 60. "计算几分钟
```

☐字符串数据处理

1.字符串移位

语法:

```
SHIFT  <c>  [BY <n> PLACES] [<modes>]  
<modes> : (1).空白, 字符串往左移一位  
          (2).LEFT, 字符串往左移 n 位  
          (3).RIGHT, 字符串往右移 n 位  
          (4).CIRCULAR: 字符串以环状方式移位
```

Example:

```
DATA  STRING(10)  VALUE 'ABCDEFGHJI'.  
SHIFT  STRING. "得到 BCDEFGHI'  
SHIFT  STRING BY 2 PLACES RIGHT. "得到 ABCDEFGH
```

2.取代字符串内容

语法:

```
REPLACE <string1> WITH <string2> INTO <c>  
将字符串 <c> 中的 <string1> 以 <string2> 来取代
```

Example:

```
DATA:  STRING(10)  VALUE 'ABCDEFGHI',  
      STR1(3)  VALUE 'DEF',  
      STR2(3)  VALUE '123'.  
REPLACE STR1 WITH STR2 INTO STRING.  
WRITE / STRING. "得到 ABC123GHI
```

3.大小写的转换

语法:

```
TRANSLATE <c> TO UPPER CASE. "转成大写  
TRANSLATE <c> TO LOWER CASE. "转成小写
```

4.在字符串中寻找部分字符串

语法:

```
SEARCH <c> FOR <str>
```

Example:

```
DATA STRING(10) VALUE 'ABCDEFGHIJ'.
```

会回存至两个变数, SY-SUBRC 和 SY-FDPOS, 若找到则 SY-SUBRC 为 0

SY-FDPOS 存开始位迭, 若找不到则 SY-SUBRC 为 4, SY-FDPOS 为 0

5.字符串长度

```
STRLEN(<c>)
```

Example:

```
INT = STRLEN('XYZABC'). "得到 6
```

```
INT = STRLEN('ABC '). "得到 3
```

6.取部分字符串

```
<f>[+<o>][<l>]
```

Example:

```
DATA T(10) VALUE 'ABCDEFGHIJ'.
```

```
WRITE / T+2(4). "得到 CDEF
```


2.5 Flow Controlling

☐比较符号

1. = 或 EQ : 等于
2. <> 或 >< 或 NE : 不等于
3. < 或 LT : 小于
4. <= 或 LE : 小于等于
5. > 或 GT : 大于
6. >= 或 GE : 大于等于
7. AND : 且
8. OR : 或
9. NOT : 非

☐条件述叙

1. IF 述叙

语法:

```
IF <Condition1>.  
    <Statement 1 >  
ELSEIF <Condition2>.  
    <Statement 2>  
ELSEIF <Condition3>.  
    <Stetement 3>  
.....  
ELSE.  
    <else Statement >  
ENDIF.
```

(1).在每个判断叙述之后要加上 .

(2).在巢状迴圈之中无法使用 ELSE 叙述, ELSE 叙述属 IF 叙述

Example:

```
IF 3 > 8.  
    WRITE / '3 is less than 8'.  
ENDIF.
```

2. CASE 叙述

语法:

```
CASE <变数 f>.
    WHEN <Value1>.
        <Statement1>
    WHEN <Value2>.
        <Statement2>
    ....
    WHEN OTHERS.
        <others Statement>
ENDCASE.
```

Example:

```
S = 'A'.
CASE S.
    WHEN 'X'.
        WRITE / 'String is X'.
    WHEN OTHERS.
        WRITE / 'String is not X'.
ENDCASE.
```

□ 迴圈叙述

1. 计次迴圈

语法:

```
DO [n TIMES] [VARYING <f> FROM <start> TO <end>].
    <loop block>
ENDDO.
```

Example:

```
DO 2 TIMES.
    WRITE / 'X'.
ENDDO.
```

执行结果:

X

X

```
DO VARYING I FROM 1 TO 10.
    S = S + I.
ENDDO.
```

```
WRITE: / , '1+2+3+...+10=', S
执行结果: 1+2+3+...+10=55
```

2. 条件迴圈

语法:

```
WHILE <Condition>.
    <Statement Block>
ENDWHILE
```

Example:

```
    I = 1.
    S=0.
    WHILE I <= 10.
        S = S+I.
        I=I+1.
    ENDWHILE.
    WRITE: / ' 1+2+3+...+10=', S.
执行结果为: 1+2+3+...+10=55
```

☐ 迴圈控制叙述

1. CONTINUE

跳至迴圈的下一次

Example:

```
DO 3 TIMES.
    IF SY-INDEX = 2.
        CONTINUE.
    WRITE / SY-INDEX.
ENDDO.
```

执行结果:

```
1
3
```

2. CHECK <Condition>

CHECK 之后条件成立才继续往下执行迴圈

Example:

```
DO 5 TIMES.
    CHECK SY-INDEX BETWEEN 2 AND 4.
    WRITE / SY-INDEX.
ENDDO.
```

执行结果:

2
3
4

3.EXIT

跳离迴圈叙述

Example:

```
DO 10 TIMES.  
  IF SY-INDEX = 4.  
    EXIT.  
  ENDIF  
  WRITE / SY-INDEX.  
ENDDO.
```

执行结果:

1
2
3

☞无穷迴圈

DO .

<Statement Block>

ENDDO.

无穷迴圈必须配合 EXIT 叙述来执行

2.6 Processing Internal Table

Internal Table 的宣告

ABAP/4 的 Internal Table 如同其它语言的数组结构, 在操作上可以有复制,删除,新增插入等功能.

1.使用 TYPE 叙述

语法:

```
TYPES <t> <type> OCCURS <n>
```

宣告一个数组 <t>, 型态为 <type>, 长度为 <n>

Example:

```
TYPES A TYPE I OCCURS 10.
```

A 是个 10 个元素的数值 Internal Table

Example:

```
TYPES: BEGIN OF LINE,  
        COL1 TYPE I,  
        COL3 TYPE I,  
        END OF LINE.
```

```
TYPES ITAB TYPE LINE OCCURS 10.
```

宣告一个 Internal Table ITAB, 总共有 10 个元素, 其 WORK AREA 名称为 LINE

2.使用 DATA 叙述

若使用 DATA 叙述来宣告 Internal Table, 可分成要不要有 HEADER LINE, HEADER LINE 就是所谓的 WORK AREA, 用在数据的存取上.

语法:

```
DATA <f> <type> OCCURS <n> [WITH HEADER LINE]
```

Example:

```
DATA VECTOR TYPE I OCCURS 10 WITH HEADER LINE.
```

3.直接宣告, 不使用 WORK AREA

语法:

```
DATA: BEGIN OF <f> OCCURS <n>,  
        <component 宣告>  
        END OF <f>.
```

Example:

```
DATA: BEGIN OF ITAB OCCURS 10,  
      COL1 TYPE I,  
      COL2 TYPE I,  
END OF ITAB.
```

如此产生的 Internal Table 不会有 Work Area, 也就是宣告时不会 Reference 其它的 Component 宣告

Append Line

语法:

```
APPEND [<wa>] TO [Initial Line To] <itab>  
[Initial Line To] 为增加一预设初值的元素
```

Example: 使用 Work Area

```
DATA: BEGIN OF LINE,  
      COL1 TYPE I,  
      COL2 TYPE I,  
END OF LINE.  
DATA ITAB LIKE LINE OCCURS 10.  
DO 2 TIMES.  
  LINE-COL1 = SY-INDEX.      "SY-INDEX 为迴圈的 Counter  
  LINE-COL2 = SY-INDEX **2.  
  APPEND LINE INTO ITAB.    "新增至 Internal Table 中  
ENDDO.  
LOOP AT ITAB INTO LINE.    "ITAB 总共有两个元素  
  WRITE: / LINE-COL1,LINE-COL2.  
ENDLOOP.
```

执行结果为:

1	1
2	4

Example: 不使用 Work Area

```
DATA: BEGIN OF ITAB OCCURS 10,  
      COL1 TYPE I,  
      COL2 TYPE I,  
END OF ITAB.  
DO 2 TIMES.
```

```

        ITAB-COL1 = SY-INDEX.
        ITAB-COL2 = SY-INDEX **2.
        APPEND  ITAB.           “新增至 Internal Table 中
    ENDDO.
    LOOP  AT  ITAB.  “ITAB  总共两个元素
        WRITE:  /  ITAB-COL1,ITAB-COL2.
    ENDLOOP.
    执行结果为:
        1          1
        2          4

```

加入另一 Internal Table 的元素

语法:

```

APPEND  LINES  OF  <itab1>  [FROM <n1>] [TO <n2>]  TO  <itab2>

```

将<itab1>的元素加入至<itab2>中, 可选取自<n1>至<n2>的范围

Example:

```

        APPEND  LINES  OF  ITAB  TO  JTAB.

```

将 ITAB 所有元素加入 JTAB 中

Collect Line

在加入新元素时将有相同 standard key(非数值字段)的数值字段汇总

语法:

```

COLLECT  [<wa>  INTO]  <itab>

```

Example:

```

DATA: BEGIN  OF  ITAB  OCCURS  3,
        COL1(3)  TYPE  C,
        COL2      TYPE  I,
    END  OF  ITAB.
    ITAB-COL1 = 'ABC'.  ITAB -COL2 = 10.
    COLLECT  ITAB.

```

```

        ITAB-COL1 = 'XYZ'.  ITAB-COL2 = 20.
    COLLECT  ITAB.

```

```

        ITAB-COL1 = 'ABC'.  ITAB-COL2 = 30.

```

```

COLLECT ITAB.    “汇总 COL2 至 COL1=ABC 的元素上
LOOP AT ITAB.
    WRITE: / ITAB-COL1,ITAB-COL2.
ENDLOOP.

```

执行结果:

```

ABC      40
XYZ      20

```

Insert Line

插入元素在指定的 Internal Table 位置之前

语法:

```

INSERT [<wa> INTO] [INITIAL LINE INTO] <itab> [INDEX <idx>]

```

Example:

```

DATA: BEGIN OF LINE,
      COL1 TYPE I,
      COL2 TYPE I,
      END OF LINE.
DATA ITAB LIKE LINE OCCURS 10.
DO 3 TIMES.
    LINE-COL1 = SY-INDEX *10.
    LINE-COL2 = SY-INDEX *20.
    APPEND LINE INTO ITAB.
ENDDO.
LINE-COL1=100.
LINE-COL2=200.
INSERT LINE INTO ITAB INDEX 2. “插入在位置 2 之前
LOOP AT ITAB INTO LINE.
    WRITE: / SY-TABIX,LINE-COL1,LINE-COL2. “SY-TABIX 为 Table 位置
ENDLOOP.

```

执行结果:

1	10	20	
2	100	200	“插入的元素
3	20	40	
4	30	60	

📖插入另一 Internal Table 元素

语法:

```
INSERT LINES OF <itab1> [FROM <n1> TO <n2>] TO <itab2> INDEX  
<idx>
```

将<itab1>的元素插入至<itab2>中, 位置在 <idx>之前, 可选取自<n1>至<n2>的范围

Example:

```
APPEND LINES OF ITAB TO JTAB INDEX 3.
```

将 ITAB 所有元素插入 JTAB 中, 位置在第三个元素之前

📖Internal Table 元素数据的读取

语法:

```
LOOP AT <itab> [INTO <wa>] [FROM <n1> TO <n2>] [WHERE <condition>]  
    <loop expression>  
ENDLOOP.
```

根据设定的范围选取原素资料, 读完后自动移往下一笔

Example:

```
LOOP AT ITAB INTO LINE WHERE COL1 >100.
```

```
    WRITE: / SY-TABIX,LINE-COL1.
```

```
ENDLOOP.
```

仅读取 COL1 > 100 的元素

📖读取 Internal Table 指定位置的元素

语法:

```
READ TABLE <itab> [INTO <wa>] INDEX <idx>
```

自指定位置 <idx> 读取元素数据

Example:

```
READ TABLE ITAB INTO LINE INDEX 5
```

读取 ITAB 的第 5 个元素数据, 放入 LINE 的字段中

📖根据字段内容寻找

语法:

```
READ TABLE <itab> INTO <wa>
```

Example:

```
ITAB-COL1 = 'ABC'.
```

```
READ TABLE ITAB INTO LINE.
```

找出 ITAB 中 COL1 字段内容是 ABC 的元素, 找到的值放入 LINE 中

若找到 SY-SUBRC 传回 0, 找不到则传回 4, <itab>必须宣告有 work area

异动元素内容

语法:

```
MODIFY <itab> [FROM <wa>] [INDEX <idx>] [TRANSPORTING <f1>...<f2>]
[WHERE <condition>]
```

TRANSPORTING <f1> ..<f2> : 指定异动的字段名称

Example:

```
LINE-COL1 = 4.
LINE-COL2 = 100.
MODIFY ITAB FROM LINE.
将目前位置元素以 LINE 的内容异动
```

Example:

```
LINE-COL1 = 10.
MODIFY ITAB FROM LINE INDEX 3 TRANSPORTING COL1.
将第三个元素的 COL1 字段异动为 10
```

Delete Lines

删除 Internal Table 的元素

语法:

```
DELETE <itab> INDEX <idx>
```

Example:

```
DELETE ITAB INDEX 4
删除第四个元素
```

加上删除条件:

```
DELETE <itab> [FROM <n1> TO <n2>] [WHERE <condition>]
```

Example:

```
DELETE ITAB FROM 3 TO 10.
删除第 3 至第 10 个元素
```

Internal Table Sorting

语法:

```
SORT <itab> [<order>] [BY <f1>] ....
```

[<order>] : 可分成递减(DESCENDING)和递增(ASCENDING), 空白表 ASCENDING

<f1>:为指定的字段

Example:

```
SORT ITAB DESCENDING BY COL2.
```

将 ITAB 根据 COL2 字段递减排序

计算数值字段总和

语法:

SUM

计算得总和存在 work area 中, 但只能存在 LOOP 叙述中

Example:

```
LOOP AT ITAB INTO LINE.  
    SUM.  
ENDLOOP.  
WRITE: / LINE-COL1,LINE-COL2.  
LINE-COL1 和 LINE-COL2 存数值总和
```

Initial Table

1. REFRESH <itab>

使用在没有 HEADER LINE 的 Internal Table 中, 清除所有元素

Example:

```
REFRESH ITAB.
```

2. CLEAR <itab>[]

使用在有 HEADER LINE 的 Internal Table 中, 清除所有元素

Example:

```
CLEAR ITAB[ ].
```

3. FREE <itab>

释放(Release) Internal Table 所占的内存空间, 用在 REFRESH 和 CLEAR 指令之后

Example:

```
FREE ITAB.
```

2.7 Processing Dictionary Table

R/3 对于存放在 Relation Database 的数据可使用 SQL 指令读取或处理, 指令种类可分成 DDL(Data Definition Language)指令, 如 CREATE, 及 DML(Data Manipulation Language)

, 如 SELECT 及 INSERT 等, 处理方式分成 OPEN SQL 及 NATIVE SQL, 前者在处理时, Database 与 Command Interpreter 间有一 Buffer 区, 如 SELECT * FROM..., 后者则直接处理数据库, 如 EXCE SQL SELECT...等, 有两个重要的系统变量:

SY-SUBRC: 传回 0 表成功执行指令, 4 表未找到符合条件数据

SY-DBCNT: 正处理的数据笔数

TABLES 指令

用于宣告程序中所使用的 tables

语法:

```
TABLES table
```

SELECT 指令

自数据库读取记录

语法:

```
SELECT <result> FROM <source> [INTO <target>] [WHERE <condition>]
      [ GROUP BY <fields>] [ORDER BY <sort order>]
```

1.以迴圈方式读取所有记录

语法:

```
SELECT [DISTINCT] * ...
```

```
.....
```

```
ENDSELECT.
```

加上[DISTINCT]会自动去除重复的记录

Example:

```
TABLES SPFLI.
```

```
SELECT * FROM SPFLI WHERE COMPANY='DELTA'.
```

```
WRITE: / PLANT,TEL.
```

```
ENDSELECT.
```

会以迴圈的方式逐笔印出符合条件的记录

2.读取单笔记录

语法:

```
SELECT SINGLE * FROM .... WHERE....
```

Example:

```
TABLES SPFLI.
```

```
SELECT SINGLE * FROM SPFLI
```

```
WHERE PLANT ='CHUNGLI' AND TEL='4526174'.
```

```
WRITE: / SPFLI-COMPANY,SPFLI-PLANT,SPFLI-TEL.
```

3.将读取的记录存放至 Work Area

语法:

```
SELECT ..... INTO <wa>
```

Example:

```
TABLES SPFLI.
```

```
DATA WA LIKE TABLES.
```

```
SELECT * FROM SPFLI INTO WA.
```

```
WRITE: / WA-COMPANY,WA-PLANT.
```

```
ENDSELECT.
```

逐笔写入 WA 工作区中

4.将读取的数据写入 Initial Table 中

语法:

```
SELECT .. INTO TABLE <itab>
```

Example:

```
TABLES SPFLI.
```

```
DATA ITAB LIKE SPFLI OCCURS 10 WITH HEADER LINE.
```

```
SELECT * FROM SPFLI INTO ITAB.
```

一次读 10 笔(Initial Table 的长度)记录存入 ITAB 中

```
SELECT .. INTO TABLE <itab> PACKAGE SIZE <n>
```

一次读取 <n> 笔记录至 <itab>中

Example:

```
TABLES SPFLI.
```

```
DATA ITAB LIKE SPFLI OCCURS 10 WITH HEADER LINE.
```

```
SELECT * FROM SPFLI INTO ITAB PACKAGE SIZE 5.
```

一次读取 5 笔记录

5.条件叙述

语法:

WHERE <condition>

(1).BETWEEN <g1> AND <g2>

在 <g1> 至 <g2> 之间的条件范围

Example:

WHERE YEAR BETWEEN 1995 AND 2000.

(2).LIKE <g>

表示条件包含的字符串

<1>._:表示一个字符

<2> % : 表示一个字符串

Example:

..WHERE NAME LIKE 'LEE%'.
条件为 NAME 字段前 3 个字符为 LEE

..WHERE MODEL LIKE "%SPS%".
条件为 MODEL 字段包含 SPS 的记录

(3).IN (<g1>....<g2>)

包含在 <g1>...<g2>的条件

Example:

..WHERE PLANT IN ('TAOYUAN','CHUNGLI','LIUTU').
条件为 PLANT 是 TAOYUAN,CHUNGLI 或 LIUTU 的记录

6.ORDER BY 叙述

指定排序的字段或顺序

(1). ..ORDER BY PRIMARY KEY.

根据 PRIMARY KEY 递增排序

(2)...ORDER BY <f1> [DESCENDING] <f2> [DESCENDING]

Example:

SELECT * FROM IM ORDER BY PART .

INSERT 指令

加入一笔记录至数据库

1.自 Work Area 工作区

语法:

INSERT INTO <database> VALUES <wa>

Example:

TABLES SPFLI.

DATA WA LIKE SPFLI.

WA-NO = '34051920'.

WA-COMPANY='DELTA'.

INSERT SPFLI VALUES WA.

将 ITAB 数据加入 SPFLI 中, 也可写成 INSERT SPFLI FROM ITAB.

SPFLI-NO='34299876'.

SPFLI-COMPANY='HP'.

INSERT SPFLI FROM SPFLI.

将 Work Area SPFLI 中的数据加入数据库档案 SPFLI 中

因 Work Area SPFLI 的结构与数据文件 SPFLI 一样, 所以也可

写成 INSERT SPFLI.

2. 自 Internal Table

语法:

INSERT <database> FROM TABLE <itab> [ACCEPTING DUPLICATE KEY]

将 <itab>中非 NULL 的数据加入 <database>中, 加上 [ACCEPTING DUPLICATE KEY]能检查不加入有重复 primary key, 若有重复则 SY-SUBRC 会传回 4

Example:

TABLES SPFLI.

DATA ITAB LIKE SPFLI OCCURS 10 WITH HEADER LINE.

ITAB-NO = '34051920'.

ITAB-COMPANY = 'DELTA'.

APPEND ITAB.

....

INSERT SPFLI FROM TABLE ITAB
ACCEPTING DUPLICATE KEY.

UPDATE 指令

异动已存在的记录内容

1. 使用 Primary Key

语法:

```
UPDATE <database> FROM <wa>
```

Example:

```
TABLES SPFLI.
```

```
DATA WA LIKE SPFLI.
```

```
WA-NO='34051920'.
```

```
WA-COMPANY='DELTA'.
```

```
UPDATE SPFLI FROM WA.
```

如 SPFLI 的 Primary Key 是 NO, 则会找到 NO='34051920' 的记录, 将其 COMPANY 字段异动为 DELTA

2.使用条件式

语法:

```
UPDATE <database> SET < f1>=<values>... WHERE <condition>
```

根据条件式异动符合条件式的记录

Example:

```
UPDATE SPFLI SET NO='34051920'
```

```
COMPANY = 'DELTA'
```

```
WHERE TEL='4526107'.
```

MODIFY 指令

根据 Primary Key 寻找数据文件中符合的记录, 若找到则更新异动, 若找不到则新增记录

语法:

```
MODIFY <database> FROM <wa>
```

Example:

```
WA-NO='34051920'.
```

```
WA-COMPANY='DELTA'.
```

```
MODIFY SPFLI FROM WA.
```

DELETE 指令

删除数据文件的记录

1.使用 Primary Key

语法:

```
DELETE <database> FROM <wa>
```

Example:

```
TABLES SPFLI.
```



```
DATA WA LIKE SPFLI.
WA-NO='34051920'.
WA-COMPANY='DELTA'.
DELETE SPFLI FROM WA.
如 SPFLI 的 Primary Key 是 NO, 则会找到 NO='34051920'
的记录, 找到后将此笔删除
```

2.使用条件式

语法:

```
DELETE FROM <database> WHERE <condition>
```

根据条件式删除符合条件式的记录

Example:

```
DELETE FROM SPFLI WHERE AREA = 'AMERICAN'.
```

DATABASE CURSOR

Database Cursor 是一个数据库暂存区, 将经 SELECT 指令读取的记录存放至此暂存区, 再由此暂存区放至 Work Area 中, 可见数据库读取的次数.

1.开启 Database Cursor

语法:

```
OPEN CURSOR <c> FOR SELECT ... WHERE <condition>
```

Example:

```
TABLES SPFLI.
DATA: WA LIKE SPFLI,
      C1 TYPE CURSOR.
OPEN CURSOR C1 FOR SELECT * FROM SPFLI
WHERE AREA = 'TAIWAN'.
```

2.读取 Database Cursor 的数据存入 Work Area

语法:

```
FETCH NEXT CURSOR <c> INTO <wa>
```

Example:

```
FETCH NEXT CURSOR C1 INTO WA.
读取下一笔 Cursor 位置的数据存入 WA, 如果已无数据可读,
SY-SUBRC 不传回 0
```

4.关闭 Database Cursor

语法:

```
CLOSE CURSOR <c>
```

Example:

```
CLOSE CURSOR C1.
```

COMMIT WORK 与 ROLLBACK WORK

要确定数据成功写入数据库, 可使用 COMMIT WORK 指令, 如:

```
COMMIT WORK.
```

相反的, 如果反悔要复原, 可使用 ROLLBACK WORK, 可复原在上个 COMMIT WORK 指令之后的数据, 如:

```
ROLLBACK WORK.
```

使用 NATIVE SQL 指令

语法:

```
EXEC SQL [PERFORMING <form>].
```

```
<statement block>] [;]
```

```
ENDEXEC.
```

Example:

```
DATA: BEGIN OF WA,
```

```
      NAME(8),
```

```
      AGE TYPE I,
```

```
      END OF WA.
```

```
DATA F1 TYPE I.
```

```
FI = 20.
```

```
EXEC SQL PERFORMING OUTPUT.
```

```
      SELECT NAME,AGE INTO :WA FROM NAME_TABLE
```

```
      WHERE AGE >= :F1.
```

```
ENDEXEC..
```

```
FORM OUTPUT.
```

```
      WRITE: / WA-NAME,WA-AGE.
```

```
ENDFORM.
```

2.8 ABAP/4 Program Module

ABAP/4 中所谓的 Module 在一般语言称之为 Subroutine, 其数据传递方式皆相似, 如 CALL BY VALUE, CALL BY REFERENCE 等, 可分成以下几个部分:

1. Macro Block
2. Include Program
3. Subroutine Program
4. Function Module

Macro Block

在程序中可以定义一段宏叙述, 并且可以传入参数, 参数符号(Placeholder)可自 &1, &2 至 &9

1. 宏的定义

```
DEFINE <macro>.  
    <宏叙述>  
END-OF-DEFINITION.
```

2. 宏的呼叫使用

```
<macro> [<p1> <p2>....]  
<p1> 为传入宏的参数值, 在参数间至少要给予一个空白
```

Example: 利用宏计算 N 次方

```
DATA RESULT TYPE I.  
DEFINE MULTI.  
    RESULT = &1 ** &2.  
    WRITE: / '&1 ^ &2 = ', RESULT.  
END-OF-DEFINITION.  
MULTI 3 4.  
执行结果为 3 ^ 4 = 81
```

Include Program

1. 在 ABAP/4 中可以使用 Include 叙述加载另一个程序的全部叙述, 通常用于共享数据项的宣告, 很类似 C 的 Include header file 的做法.

语法:

INCLUDE <include program file>

Example:

程序 YStart 的内容如下:

```
***INCLUDE YSTART.
```

```
WRITE: / 'User Name = ',SY-UNAME.
```

```
WRITE:/ 'Host Server = ',SY-HOST.
```

另一程序如下:

```
PROGRAM YTEST1.
```

```
INCLUDE YSTART. “载入 YSstart 的所有内容
```

执行结果:

```
User Name = MIS-CHOU
```

```
Host Server = deidv01
```

2.Global 变量宣告应用

语法:

```
DATA: BEGIN OF COMMON PART [<name>],
```

```
<data 宣告>
```

```
END OF COMMON PART [<name>]
```

此常使用在 Include 的档案中, 如

```
***INCLUDE INCOMMON.
```

```
DATA: BEGIN OF COMMON PART NUMBERS,
```

```
MID(8),
```

```
MNUM TYPE I,
```

```
END OF COMMON PART NUMBERS.
```

Subroutine Procedure

在 ABAP/4 Subroutine 的呼叫可分成 Internal Call 和 External Call, 前者撰写在程序中, 后者存在另一程序中, 通常为专存放 Subroutine 的公用程序集, 可提供给不同的程序呼叫.

1.Subroutine 的宣告

```
FORM <subr> [<pass>].
```

```
<subroutine statement block>
```

```
ENDFORM.
```

2.呼叫的方法

(1).Internal Call

语法:

```
PERFORM <subr> [<pass>]
```

Example:

```
NUM1 = 100.  NUM2 = 200.
PROFORM  ADD.
FORM  ADD.
    SUM = NUM1 + NUM2.
    WRITE: / 'NUM1 + NUM2 = ',SUM.
ENDFORM.
执行结果:  NUM1 + NUM2 = 300
```

(2).External Call 另一程序

语法:

```
PERFORM  <subr>(<prog>) [<pass>] [IF FOUND]
```

<subr>:子程序名称

<prog>:存放子程序的程序名称

IF FOUND: 找到才执行

Example:

```
PROGRAM  FORMPOOL.
FORM  HEADER.
    WRITE: / 'USER NAME: ',SY-UNAME.
ENDFORM.
在程序中呼叫 HEADER 子程序
PROGRAM  YTEST1.
PERFORM  HEADER(FORMPOOL) IF  FOUND.
```

(3).External Call 另一专存放子程序的程序文件

语法:

```
PERFORM  (<fsubr>) [IN  PROGRAM (<fprog>) [<pass>] [IF FOUND]
```

Example:

存放子程序的程序文件

```
PROGRAM  FORMPOOL.
FORM  SUB1.
    WRITE: / 'USER NAME:',SY-UNAME.
ENDFORM.
FORM  SUB2.
    WRITE: / 'HOST SERVER:',SY-HOST.
ENDFORM.
在程序中呼叫 FORMPOOL 中的 SUB2 子程序
SUBNAME = 'SUB2'.
```

```

PROGNAME = 'FORMPOOL'.
PERFORM (SUBNAME) IN PROGRAM (PROGNAME) IF FOUND

```

参数值的传递

在 ABAP/4 中参数的传递可分成

1. Call By Reference:

传参数时将数据的存放地址(address)传至参数中, 也就是子程序中的参数变量与外部实际变量共享地址内的值, 又称为 Call By Address, 若在子程序中地址中的值改变了, 外部实际变量的值也会跟着改变.

语法:

```

FORM <subr> [USING <f1> <f2>...] [CHANGING <f1>...]
PERFORM <subr> [USING <f1> <f2>...] [CHANGING <f1>...]

```

Using 之后接在子程序中不会改变的变量, CHANGING 接会改变值的变量
但实际上 USING 之后的参数在子程序中也可将值改变

Example:

```

SUM = 0.
NUM1 = 100. NUM2=200.
PERFORM ADD USING NUM1 NUM2 CHANGING SUM.
WRITE: / NUM1,NUM2,SUM " SUM 由 0 变成 300
FORM ADD USING NUM1 NUM2 CHANGING SUM.
    SUM = NUM1 + NUM2.
ENDFORM.

```

执行结果:

```

100  200  300

```

2. Call By Value

传参数时将数据的值复制一份至另一地址中, 所以在子程序中参数变量值改变, 并不会影响外部实际变数的值.

语法:

```

FORM <subr> USING VALUE(<f1>...)

```

使用 VALUE(<f1>)表示 <f1>是 Call By Value 的传递

```

PERFORM <subr> USING <f1>

```

Example:

```

SUM = 0.
NUM1 = 5.
PERFORM MULTI USING NUM1 CHANGING SUM.

```

```

WRITE: / NUM1,SUM  "NUM1 值还是 5, SUM 由 0 变成 120
FORM MULTI USING VALUE(NUM1) CHANGING SUM.
SUM = 1.
WHILE NUM1 > 1
SUM = SUM * NUM1.
NUM1 = NUM1 - 1.
ENDWHILE..
ENDFORM.

```

执行结果:

```

5    120

```

3.Call By Value and Return Result

传入参数值的方式同 Call By Value, 但在子程序结束执行时会将传入的参数值复制一份传回给外部实际变数.

语法:

```

FORM ..... CHANGING VALUE(<f1>)
PERFORM .... CHANGING .... <f1>

```

Example:

```

SUM = 0.
NUM1 = 100. NUM2=200.
PERFORM ADD USING NUM1 NUM2 CHANGING SUM.
WRITE: / NUM1,NUM2,SUM  " SUM 由 0 变成 300
FORM ADD USING NUM1 NUM2 CHANGING VALUE(S).
S = NUM1 + NUM2.
WRITE: / NUM1,NUM2,SUM  "得到结果为 100 200 0
ENDFORM.

```

执行结果:

```

100    200    0    "在子程序中 SUM 值尚未改变
100    200    300   "返回程序时, 将变量 S 的值复制给 SUM
                        "所以 SUM 值变成 300

```

📖 Subroutine 的控制

5.CHECK <Condition>

CHECK 之后条件成立才继续往下子程序叙述

Example:

```

PERFORM SUB1.
    CHECK NUM1 < 10.
    WRITE / NUM1.
    NUM1 = NUM1 + 1.
ENDFORM.

```

6.EXIT

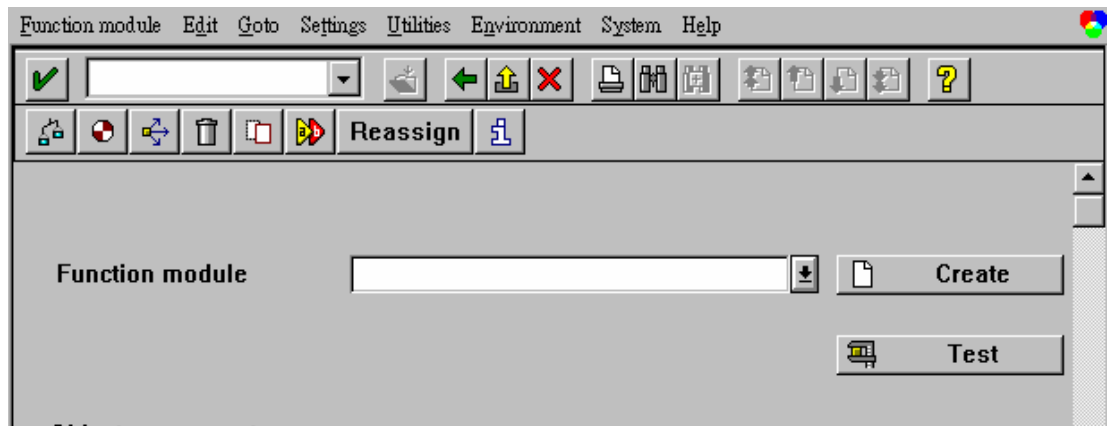
强迫结束子程序执行，返回上一层程序叙述

Function Module

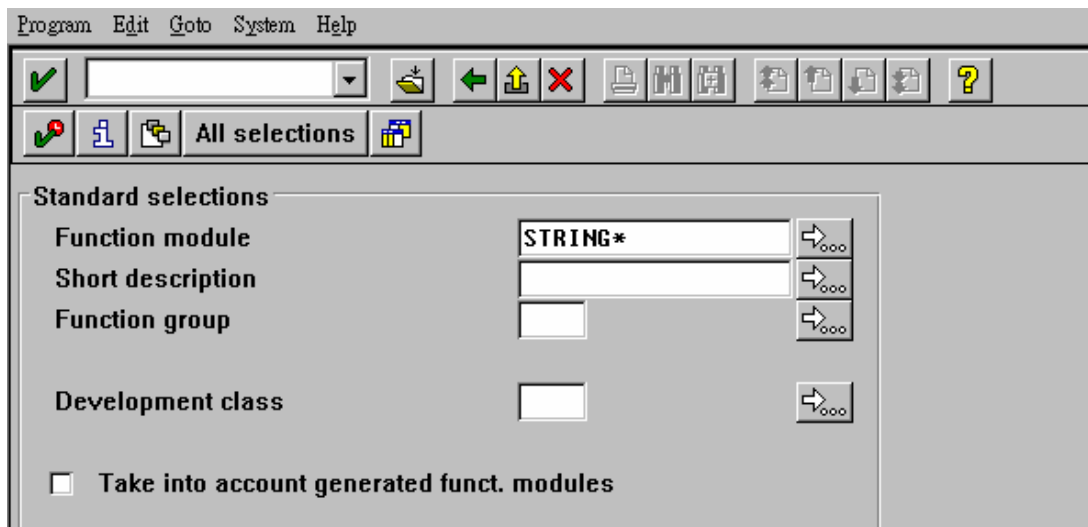
在 ABAP/4 中的 Function Module 是储存在一个函式库中(library)，系统提供很多内设的 Function Module 供程序中呼叫，也可以自行增加自己的 Function Module.

1.叫出已存在的 Function Module

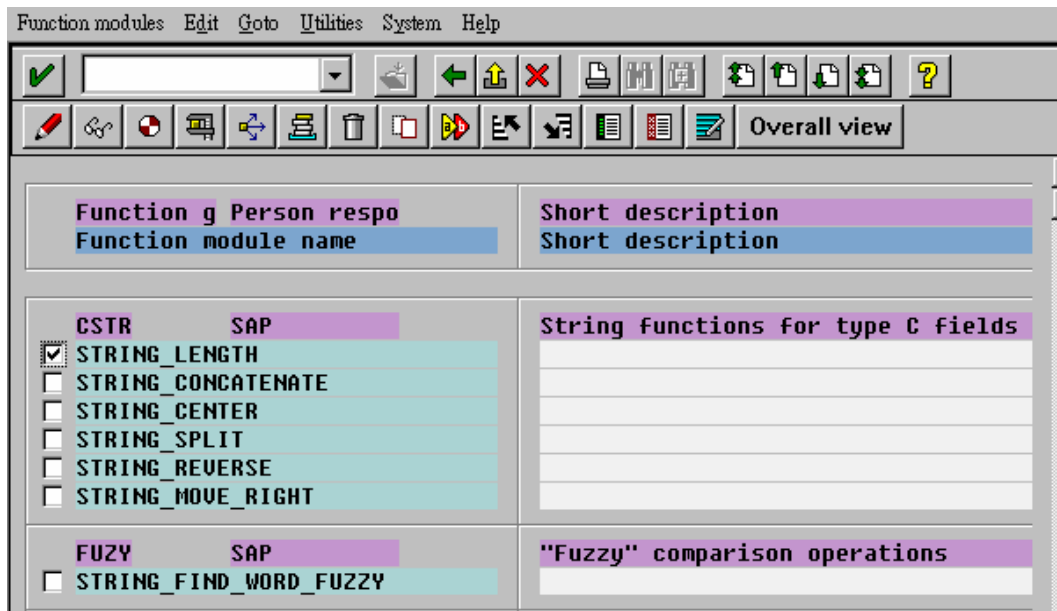
(1).在 ABAP/4 Development Workbench 画面中选择 “Function library”，可见以下画面：



(2).选择“Utility”中的“Find”中输入要寻找的 Function Module 名称，如输入 STRING*，为找出前五个字符为 STRING 的 Function Module:



(3).输入后按下左上的 Execute, 可见以下画面:



(4).选择要查询或修改的 Function 名称, 如选择 STRING_LENGTH, 如要查询可选择

“Display”, 可见 Function Module 之各项参数设定:

<1>.Import Parameter: 传入的参数名称, 但实际在程序中使用刚好与 Export 相颠们

<2>.Export Parameter: 传回的参数名称, 程序中变成 Import 的使用

<3>.Changing Parameter:使用 Call By Value and Return Result 方法的参数

<4>.Table Parameter: 使用的 Initial Table 参数

<5>.Exceptions: 错误处理参数

如 STRING_LENGTH 为一传回字符串长度的函数, 其设定的参数如下:

Import : String 传入一字符串字符串数据

Export: Length 传回的字符串长度值

(5).要查看程序内容可按下 “ Source Code”, 显示其程序内容如下:

```
function string_length.  
  
length = strlen(string).  
  
endfunction.
```

(6).函数的呼叫

语法:

```
CALL FUNCTION <module>  
  IMPORTING F1=a1....  
  EXPORTING F1=a1....  
  CHANGING F1=a1...  
  TABLES F1=a1...  
  EXCEPTIONS F1=a1...
```

Example:

```
DATA: TEXT(20),  
      LEN TYPE I.  
TEXT = 'ABCDEFGHJIJ'.  
CALL FUNCTION 'STRING_LENGTH'  
  EXPORTING STRING = TEXT  
  IMPORTING LENGTH = LEN.  
WRITE / LEN.
```

注意 EXPORTING 与 IMPORTING 刚好颠倒,
执行所得结果为 10

4. 屏幕输入命令

在 ABAP/4 中要自屏幕输入变量数据的内容, 使用的命令是 PARAMETERS 及, SELECTION-OPTIONS:

1. PARAMETER: 输入一个变量或字段内容
2. SELECTION-OPTIONS: 使用条件筛选画面来输入数据

📖.PARAMETERS 指令

基本的输入命令, 类似如 BASIC 的 INPUT 命令, 但无法使用 F 格式(浮点数)

语法:

```
PARAMETERS <p> [DEFAULT <f>] [LOWER CASE]
               [OBLIGATORY] [AS CHECKBOX]
               [RADIOBUTTON GROUP <rad>]
```

Example:

```
PARAMETERS: NAME(8),
              AGE TYPE I,
              BIRTH TYPE D.
```

执行结果:

NAME	LILY
AGE	32
BIRTH	08/05/1966

在日期的输入格式上为 MM/DD/YY, MM/DD/YYYY, MMDDYY
或 MMDDYYYY, 如输入 020165 表 1965 年 02 月 01 日, 与 02/01/65
的输入是一样的, 日期输入范围为公元 1950 年至 2049 年

1. DEFAULT

设定输入的默认值

Example:

```
PARAMETERS: COMPANY(20) DEFAULT 'DELTA',
              BIRTH TYPE D DEFAULT '19650201'.
```

2.LOWER CASE

ABAP/4 预设是将字符串输入值自动转换为大写, 加上此参数会将输入的数据
转成小写,

3.OBLIGATORY

强制要求输入, 屏幕上会出现一个 ?, 使用者必须要输入才可.

4.AS CHECKBOX

输入 CHECKBOX 的格式

Example:

```
PARAMETERS: TAX AS CHECKBOX DEFAULT 'X',  
            NTD AS CHECKBOX.
```

执行结果:

A screenshot of a form with a light gray background. It contains two checkboxes. The first checkbox is checked and is labeled 'TAX'. The second checkbox is unchecked and is labeled 'NTD'.

5. RADIOBUTTON GROUP <rad>

输入 RADIO BUTTON GROUP 的方式

Example:

```
PARAMETERS: BOY RADIOBUTTON GROUP SEX DEFAULT 'X',  
            GIRL RADIOBUTTON GROUP SEX.
```

执行结果:

A screenshot of a form with a light gray background. It contains two radio buttons. The first radio button is selected and is labeled 'BOY'. The second radio button is unselected and is labeled 'GIRL'.

SELECTION-OPTIONS

条件筛选检查条件输入画面指令，输入条件后可配合 SELECT 指令自 TABLE 读取符合条件的数据，直接执行或放入 Internal Table 中，条件有四个参数：

1. SIGN:

I: 表筛选条件符合的资料

E: 表筛选条件不符合的资料

2. OPTION: 比较的条件符号

EQ(等于),NE(不等于),GT(大于),LE(小于),CP(包含),NP(不包含)

3. LOW: 最小值

4. HIGH: 最大值

语法:

```
SELECTION-OPTIONS <check-option> FOR <table-field>
```

Example:

```
TABES SPFLI.
```

```
SELECTION-OPTIONS AIRLINE FOR SPFLI-CONNID.
```

将条件的输入值存放入 AIRLINE, 筛选选择为 SPFLI 中的
CONNID 字段

执行结果:

可直接输入起始范围或按下选择画面，输入完后按下左上角的执行键

条件输入选择画面

1. 自 Table 中选取

按下输入项的右边往下箭头，叫出 Table 中数据项，选取开始和结束的范围

2. Selection Options

按下“Selection options”按键，输入 Option 及 Sign 参数内容，屏幕如下：

3. Multi-Options 输入

按下最右边的 Multi-Options 输入键，输入条件选取的范围，画面如下：

条件输入完后按下“Copy”按键

📖 改变条件输入格式

1. DEFAULT <begin> TO <end>

设定开始结束范围输入默认值

Example:

```
SELECTION-OPTION AIRLINE FOR SPFLI-CONNID  
DEFAULT '2042' TO '4555'.
```

2. NO-EXTENSION

设定不要 Multi-Option 输入画面

3. NO INTERVALS

设定不要区间范围输入画面

4. LOWER CASE

输入转换成大写

5. OBLIGATORY

强制要求输入

📖 配合 SELECT 命令

条件输入完后要将符合条件的数据筛选出来, 可配合使用 SELECT 指令

1. 使用 WHERE <条件式>

Example:

```
SELECT-OPTIONS AIRLINE FOR SPFLI-CONNID.  
SELECT * FROM SPFLI WHERE CONNID IN AIRLINE.  
WRITE: / CONNID, FROMCITY, TOCITY.  
ENDSELECT.
```

2. 使用 CHECK 参数

Example:

```
SELECT-OPTIONS AIRLINE FOR SPFLI-CONNID.  
SELECT * FROM SPFLI.  
CHECK AIRLINE.  
WRITE: / CONNID, FROMCITY, TOCITY.  
ENDSELECT.
```

3. 使用 IF ... IN 叙述

Example:

```
SELECT-OPTIONS AIRLINE FOR SPFLI-CONNID.  
SELECT * FROM SPFLI.  
IF SPFLI-CONNID IN AIRLINE.  
WRITE: / CONNID, FROMCITY, TOCITY.
```

ENDIF
ENDSELECT.

SELECTION-SCREEN

1.产生空白列

语法:

SELECTION-SCREEN SKIP [<n>]

Example:

SELECTION-SCREEN SKIP 2.
产生两列空白列

2.产生底线

语法:

SELECTION-SCREEN ULINE / <pos>(length)

Example:

SELECTION-SCREEN ULINE /10(30).
自第 10 格开始产生长度 30 的底线

3.印出备注说明

语法:

SELECTION-SCREEN COMMENT / <pos>(length) <name>

Example:

REMARK = 'Pls enter your name'.
SELECTION-SCREEN COMMENT /10(30) REMARK.

4. 同一列中输入数个数据项

语法:

SELECTION-SCREEN BEGIN OF LINE.

.....

SELECTION-SCREEN END OF LINE.

Example:

SELECTION-SCREEN BEGIN OF LINE.
SELECTION-SCREEN POSITION 20.
PARAMETERS NAME(10).
SELECTION-SCREEN POSITION 40.
PARAMETERS BIRTH TYPE D.
SELECTION-SCREEN END OF LINE.
在 20 格输入 NAME 内容, 40 格输入 BIRTH 的内容

5. 绘出 BLOCK PANEL

语法:

```
SELECTION-SCREEN BEGIN OF BLOCK <block>
    [WITH FRAME [TITLE <title>].
```

.....

```
SELECTION-SCREEN END OF BLOCK <block>.
```

Example:

```
SELECTION-SCREEN BEGIN OF BLOCK RADIO
    WITH FRAME .
    PARAMETER R1 RADIOBUTTON GROUP GR1.
    PARAMETER R2 RADIOBUTTON GROUP GR1.
    PARAMETER R3 RADIOBUTTON GROUP GR1.
SELECTION-SCREEN END OF BLOCK RADIO.
```


5.Standard Report

一个典型的报表程序是由许多的程序区块(Code Block)所组成，在区块间最好能加上一些说明以利程序可读性，一个典型的报表程序格式如下：

* **PROGRAM SOURCE HEADER** : 说明程序名称及目的

* Program Name:

* Description:

* Date/Author:

* Table Update:

* Special Logic:

* Include:

* -----

* **MODIFICATION LOG** : 程序修改更新记录

* -----

* ChangeDate Programmer Request Description

* =====

=====

* NEW PROGRAM

* -----

* **REPORT NAME** : 宣告程序名称及报表格式,

* -----

REPORT Z_____

NO STANDARD PAGE HEADING

MESSAGE-ID __ " 所使用的 MESSAGE

LINE-COUNT __ " 每页报表列数

LINE-SIZE __. " 每页报表宽度

* **TABLE DESCRIPTION** : 宣告程序会使用的 TABLE

* -----

TABLES:

* **DATA** : 宣告程序所使用的变量及自定型态

* -----

TYPES:

DATA:

* **SELECTION SCREEN / OPTION / PARAMETER** : 屏幕输入报表筛选条件

* -----

SELECTION-SCREEN BEGIN OF BLOCK _____

SELECT-OPTIONS:

SELECTION-SCREEN END OF BLOCK _____

* **INITIALIZATION** : 启动程序开始执行, 如 **SELECT-OPTION** 及 **PARAMETER**

* -----
INITIALIZATION.

INCLUDE ____.

* **AT START SELECTION** : 输入结束后启动的区块, 如按下<F8>

* -----
START-OF-SELECTION.

SET PF-STATUS ____ . " 指定报表执行时所用的 GUI-STATUS 名称

PERFORM READ_DATA.
PERFORM PROCESS_DATA.
PERFORM PRINT_DATA.
PERFORM PRINT_SUMMARY.

* **AT USER Command** : 执行在 **GUI-STATUS** 中自定的命令

* -----
AT USER_COMMAND.

* **AT LINE SELECTION** : 由在报表中按下<F2>或 **Double-Click** 启动

* -----
AT LINE-SELECTION.

* **TOP OF PAGE** : 每页开始打印时执行, 用于定义报表表头

* -----

* **END OF PAGE** : 报表打印完最后一页后启动

* -----
END-OF-PAGE

* **END OF SELECTION** : 在结束打印数据后启动, 如可用来印出 **USER** 输入的条件

* -----
END-OF-SELECTION.

INCLUDE ____

* **FORM** : 撰写程序中所使用到的子程序

* -----

* **Read Data** : 自 **TABLE** 读取数据放入 **Internal Table**

* -----

```

FORM READ_DATA.
  SELECT * FROM _____
        INTO _____
        WHERE _____.
  IF SY-SUBRC = 0.

    ENDIF.
    APPEND _____. " 增加 Internal Table 元素
  ENDSELECT.

ENDFORM.

* Process Data : 处理 Internal Table 的数据, 如排序及汇总
* _____
FORM PROCESS_DATA.

ENDFORM.

* Print Data : 依序输出 Internal Table 的数据
* _____
FORM PRINT_DATA.

ENDFORM.

* Print Summary : 印出数值资料加总
* _____
FORM PRINT_SUMMARY.

ENDFORM.

* Include Program : 列出所含入的其它程序 source code, 如子程序
* _____
INCLUDE _____
INCLUDE _____

```

6. Write BDC Program

BDC Program (Batch Data Communication Program) 是 ABAP/4 用来加载数据异动 SAP 数据库的方法, 先将要输入的数据存在 BDC Table 中, 使用 CALL TRANSACTION 指令呼叫 R/3 输入画面, 将输入所需数据项自 BDC Table 中依序放入, 最后送出按键句柄, 如 /11 表按下 <F11> 存盘, 此方法用在自不同系统转入 R/3 系统之数据转换,

(Data Migrarion), 或者也可使用在 Drill-Down 报表的撰写方式中。

6.1 Screen of Transaction Status

当 user 使用 SAP R/3 来输入数据时, SAP R/3 使用 Transaction 来执行工作的执行, 每个 Transaction 都会包含一些不同的 Screen , 每个 Screen 被定义为一个程序名称及一个 Screen Nmbler, 在执行 SAP R/3 时选择 “System”中的“Status”可看到如下的画面:

The screenshot shows the 'System: Status' window with the following data:

Usage data		Previous logon	
Client	035	06/10/1998	10:52:22
User	MIS-PR10	Logon	13:58:13
Language	E	System time	18:51:22

SAP data		SAP System data	
Repository data		SAP Release	
Transaction	VA03	31H	
Program (screen)	SAPMV45A	Installation number	7320024305
Screen number	400	License expiration	12/31/9999
Program (GUI)	SAPMV45A	Patch	
GUI status	U	Type	HOT
		Name	SAPKH31H19
		Status	I

如图: Transaction Code 是 VA03, 程序名称是 SAPMV45A, Screen Number 是 400

6.2 BDC Table

BDC Table Structure

BDC Table 是一个 Structure, 用来存放要放入输入画面的数据, 包含有以下的字段:

Field Name	Type	Description
Program	Char(8)	Program name of Transaction

Dynpro	Char(4)	Screen number of Transaction
Dynbegin	Char(1)	Indicator for new Screen
Fnam	Char(35)	Name of Database Field from Screen
Fval	Char(80)	Value to Submit to Field

可在程序开始之初宣告一个 Internal Table 使用 BDCDATA 的 Structure:

```
DATA BEGIN OF INT_BDC OCCURS 0.
    INCLUDE STRUCTURE BDCDATA.
DATA END OF INT_BDC.
```

如在以上的画面中要输入 VBAK-KUNNR 及 VBAK-NAME1 两个字段的內容，分别要填入 '34051920' 及 '台达电子'，其 BDC Table 内容如下:

Program	Dynpro	Dynbegin	Fnam	Fval
SAPMV45A	0300	X		
			VBAK-KUNNR	34051920
			VBAK-NAME1	台达电子
			BDC_OKCODE	/nn

BDC_OKCODE 字段的值就是结束输入时所要按下的键盘句柄，

☐ 常用的 BDC Okcode:

<u>OKCODE</u>	<u>Description</u>
/nn	Function Key nn
/00	Enter
/8	F8, Continue or Execute
/11	F11, Post
%EX	Exit
BACK	F3, Back Previous Screen
DLT	Delete
PICK	Double Click
SAVE	F11, Save

☐ 存入 BDC Table 的资料

首先我们需建立两个子程序，BDC_SCREEN 是来存入 Program，Dynpro 和 Dynbegin 三个字段，也就是输入画面的程序名称及 Screen Number，BDC_FIELD:用来存入 Fnam 及 Fval 两个字段，也就是输入画面所需填入的各个字段内容，程序内容如：

*** Add BDC Screen Field Data**

```
FORM BDC_SCREEN TABLES P_BDC STRUCTURE BDCDATA
    USING P_PROGRAM P_SCREEN.
    CLEAR P_BDC.
    P_BDC-PROGRAM = P_PROGRAM.
    P_BDC-DYNPRO = P_SCREEN.
    P_BDC-DYNBEGIN = 'X'.
    APPEND P_BDC.
ENDFORM.
```

*** Add BDC Field Data**

```
FORM BDC_FIELD TABLES P_BDC STRUCTURE BDCDATA
    USING P_NAME P_VALUE.
    CLEAR P_BDC.
    CASE P_VALUE.
        WHEN ".
        WHEN OTHERS.
            P_BDC-FNAM = P_NAME.
            P_BDC-FVAL = P_VALUE.
            APPEND P_BDC.
    ENDCASE.
ENDFORM.
```

接下来在程序中去呼叫这两个子程序来填入数据，程序叙述如下：

```
PERFORM BDC_SCREEN TABLES INT_BDC
    USING 'SAPMV45A' '0300'.
PERFORM BDC_FIELD TABLES INT_BDC:
    USING 'VBAK-KUNNR' '34051920',
    USING 'VBAK-NAME1' '台达电子',
```

```
USING 'BDC_OKCODE' '00'.      " Save and end
```

6.3 Call Transaction 指令

将数据依序填入 BDC Table 后要写入 R/3 的 Database 时要使用 CALL TRANSACTION 指令，指令格式如下：

```
CALL TRANSACTION WITH <tcode>
                  USING <BDC Table>
                  MODE <Display Mode>
```

<Display Mode> 可分成：

- A Show all Screen
- E Show only Screen with Error
- N Show no Screen

如：

```
CALL TRANSACTION 'VA03'
                USING INT_BDC
                MODE 'E'.
```

6.4 Example for BDC Program

例如我们想经由 BDC Program 将 Sales Employee 的数据由 ASCII Text File 去更新 R/3 数据库 Sales Employee 的 Personal Data，会有以下两个画面要输入：

The screenshot shows the 'Maintain Sales Representative' dialog box in SAP. The title bar reads 'Maintain Sales Representative'. The menu bar includes 'Sales personnel', 'Edit', 'Goto', 'Environment', 'Auxiliary functions', 'System', and 'Help'. The toolbar contains various icons for navigation and actions. The main area has a 'Personnel number' field with the value '242147'. Below this, there are two sections: 'Sales personnel' and 'Period'. In the 'Sales personnel' section, there are radio buttons for 'Events', 'Addresses', 'Personal Data' (which is selected), and 'Organizational Assignment'. There is also an 'Infotype' field with 'Personal Data' and an 'STy' field. In the 'Period' section, there are radio buttons for 'Period' (selected), 'Today', 'From' (with a date field), 'To' (with a date field), 'To current date', 'From curr.date', and 'All'. A 'Choose' button is located at the bottom right of the 'Period' section.

Program Name 是 SAPMP50A, Screen Number 2042, Transaction Code 是 PAL3

在 Screen 1000 要填入的资料是

Description	Field Name	Value
Personal Number	RP50G-PERNR	242147
Info Type	RP50G-CHOIL	Personal Data

在 Screen 2042 要填入的资料是

Description	Field Name	Value
From Date	P002-BEGDA	01/13/1969
To Date	P002-ENDDA	12/31/9999
Last Name	P002-NACHN	Dora
First Name	P002-VORNA	Cheng

欲转入的 ASCII Text File 檔名是 empoly.txt, 其格式为

Description	Length	Position	Value
Personal Number	10	1	242147
Info Type	20	11	Personal Data
From Date	10	31	01/13/1969
To Date	10	41	12/31/9999
Last Name	20	61	Dora
First Name	20	81	Cheng

程序如下:

```
* PROGRAM SOURCE HEADER
* Program Name: ZTBDC00
* Description: Change Sales Employee Data Using BDC Program
* Date/Author: 1998/06/10 周庆日
* Table Update:
* Special Logic:
* Include:
* -----
* MODIFICATION LOG
* -----
```


* ChangeDate	Programmer	Request	Description
* 1998/06/10	Chou		NEW PROGRAM

* REPORT NAME

```

REPORT ZTBDC00
      NO STANDARD PAGE HEADING
      MESSAGE-ID ZZ
      LINE-COUNT 60
      LINE-SIZE 80.

```

* DATA

```

TABLES: RP50G,P002.
DATA: BEGIN OF INT_BDC OCCURS 0.
      INCLUDE STRUCTURE BDCDATA.
DATA: END OF INT_BDC.

```

```

* Work Internal Table
DATA: BEGIN OF IN_REC,
      PERNR LIKE RP50G-PERNR,
      CHOIL LIKE RP50G-CHOIL,
      BEGDA LIKE P002-BEGDA,
      ENDDA LIKE P002-ENDDA,
      NACHN LIKE P002-NACHN,
      VORNA LIKE P002-VORNA,
      END OF IN_REC.

```

```

DATA: P_FILE(30) VALUE 'empoly.txt'. "ASCII Text File Name

```

* INITIALIZATION

```

INITIALIZATION.

```

* AT START SELECTION

```

START-OF-SELECTION.
  PERFORM READ_DATA.
  PERFORM PROCESS_DATA.

```

* END OF SELECTION

```

END-OF-SELECTION.
  PERFORM CLEAN_UP.

```

* FORM

*** Read Data OPEN ASCII Text File**

* -----
FORM READ_DATA.
OPEN DATASET P_FILE FOR INPUT IN TEXT MODE.

IF SY-SUBRC NE 0.
MESSAGE E999 WITH 'Cannot Open ' P_FILE.
ENDIF.

ENDFORM.

*** Process Data**

* -----
FORM PROCESS_DATA.
DO.
READ DATASET P_FILE INTO IN_REC. "Add Data To Internal Table
IF SY-SUBRC NE 0.
EXIT.
ENDIF.

PERFORM BUILD_BDC. "Add to BDC Table
PERFORM SUBMIT_BDC. "Call Transaction Code To Change Data

ENDDO.

ENDFORM.

* -----

*** Add Data to BDC Table**

FORM BUILD_BDC.
PERFORM BDC_SCREEN TABLES INT_BDC
USING 'SAPMP50A' '1000'.
PERFORM BDC_FIELD TABLES INT_BDC:
USING 'RP50G-PERNR' IN_REC-PERNR,
USING 'RP50G-CHOIL' IN_REC-CHOIL,
USING 'BDC_OKCODE' '/00'.

PERFORM BDC_SCREEN TABLES INT_BDC
USING 'SAPMP50A' '2042'.
PERFORM BDC_FIELD TABLES INT_BDC:
USING 'P0002-BEGDA' IN_REC-BEGDA,
USING 'P0002-ENDDA' IN_REC-ENDDA,
USING 'P0002-NACHN' IN_REC-NACHN,
USING 'P0002-VORNA' IN_REC-VORNA,
USING 'BDC_OKCODE' '/11'.

ENDFORM.

* -----

* Call Transaction Code to Execute Change Data
FORM SUBMIT_BDC.

```

CALL TRANSACTION 'PAL3' USING INT_BDC
MODE 'N'.
REFRESH INT_BDC. "将 BDC Table 资料清除
ENDFORM.

```

```

* -----
* Close ASCII Text File
FORM CLEAN_UP.
CLOSE DATASET P_NAME.
IF SY-SUBRC NE 0.
MESSAGE E999 WITH 'Cannot Close ' P_FILE.
ENDIF.

```

```

ENDFORM.

```

```

* -----

```

* Add BDC Screen Field Data

```

* -----

```

```

-
FORM BDC_SCREEN TABLES P_BDC STRUCTURE BDCDATA
USING P_PROGRAM P_SCREEN.
CLEAR P_BDC.
P_BDC-PROGRAM = P_PROGRAM.
P_BDC-DYNPRO = P_SCREEN.
P_BDC-DYNBEGIN = 'X'.
APPEND P_BDC.
ENDFORM.

```

* Add BDC Field Data

```

* -----

```

```

FORM BDC_FIELD TABLES P_BDC STRUCTURE BDCDATA
USING P_NAME P_VALUE.
CLEAR P_BDC.
CASE P_VALUE.
WHEN ".
WHEN OTHERS.
P_BDC-FNAM = P_NAME.
P_BDC-FVAL = P_VALUE.
APPEND P_BDC.
ENDCASE.
ENDFORM.

```

7.OLE Automation

OLE(Object Linking and Embendding)是由 Microsoft 所制定的标准,可以让一个程序经由此去整合另一程序的对象(Object),OLE Automation 是其中的一部分,经由既定的语法格式去产生连结的对象数据,可以分成 Objects of Application、Call its Methods 和 Set and Get Object Properties。

Using OLE Automation from ABAP/4 Program

在 ABAP/4 中提供了许多的 OLE Automation 接口,让 ABAP/4 程序可经由接口去整合其它应用软件(如 WORD、EXCEL 等)至 R/3 系统中,要知道有那些的接口可使用,在 WorkBench 中选择 Development->Programming Environ->OLE2->OLE2 Object Browser 中去查询,至于更详尽的数据则需参考 Microsoft 发行的 OLE 2.0 Reference 或 MSDN 相关的资料。。

Object of Application

要使用 OLE Automation 之前需在 ABAP/4 程序中先产生对象,指令是

```
CREATE OBJECT <对象名称> <对象函式库>
```

如要建立一个 EXCEL 5.0 的对象名称 application:

```
INCLUDE OLE2INCL.
```

```
DATA : APPLICATION TYPE OLE2_OBJECT.
```

```
CREATE OBJECT APPLICATION 'Excel.application'.
```

Object Properties

对象产生后就可以设定其属性(Property)了,指令是

```
SET PROPERTY OF <对象名称> <属性> = <Value>
```

如设定对象 APPLICATION 可在屏幕上显示:

```
SET PROPERTY OF APPLICATION 'Visible' = 1.
```

如果想获得目前对象属性的值, 指令如下:

```
DATA: VISIBLE TYPE I.
```

```
GET PROPERTY OF APPLICATION 'Visible' = VISIBLE.
```

Calling Method

指令格式:

```
CALL METHOD OF <对象名称> <Method> = <Value>
```

如要存放 R/3 的数据至 EXCEL 的存格(CELL)中, 在 ABAP/4 的指令如下:

```
INCLUDE OLE2INCL.
```

```
DATA: APPLICATION TYPE OLE2_OBJECT,  
      WORKBOOK TYPE OLE2_OBJECT,  
      SHEET TYPE OLE2_OBJECT,  
      CELLS TYPE OLE2_OBJECT.
```

```
CREATE OBJECT APPLICATION 'Excel.application'.
```

```
SET PROPERTY OF APPLICATION 'Visible' = 1.
```

* 宣告一 EXCEL 工作底稿档(WORKBOOK)

```
CALL METHOD OF APPLICATION 'Workbooks' = WORKBOOK.
```

* 增加一新的工作表(SHEET), 编号是 1 号

```
CALL METHOD OF WORKBOOK 'Add'.
```

```
CALL METHOD OF APPLICATION 'Worksheets' = SHEET  
      EXPORTING #1 = 1.
```

* 设定此工作表开启使用

```
CALL METHOD OF SHEET 'Activate'.
```

Free Object

对象使用完后必须自 Memory 中释放，指令是

```
FREE OBJECT <物件>
```

如:

```
FREE OBJECT APPLICATION.
```

将对象 APPLICATION 自 Memory 释放

Using OLE to Add Data

以要将 KNA1(Customer General Data)中的资料存入 EXCEL 的存格(CELL)为例:

```
FORM FILL_SHEET.
DATA: ROW_MAX TYPE I VALUE 256,
      ROWS     TYPE I VALUE 1.
      INDEX     TYPE I.
FIELD-SYMBOLS: <NAME>.
SELECT * FROM KNA1.
  ROWS = ROWS + 1. "至第 ROWS 列
  INDEX = ROW_MAX * ( ROWS - 1 ) + 1.
  DO 10 TIMES. "如要每一列放入 10 个存格的数据
    ASSIGN COMPONENT SY-INDEX OF STRUCTURE KNA1
      TO <NAME>.
    CALL METHOD OF SHEET 'Cells' = CELLS
      EXPORTING #1 = INDEX.
    SET PROPERTY OF CELLS 'Value' = <NAME>.
    ADD 1 TO INDEX.
  ENDDO.
ENDSELECT.
```

完整的程序

```
REPORT EXCEL.
INCLUDE OLE2INCL.
DATA: APPLICATION TYPE OLE2_OBJECT,
      WORKBOOK    TYPE OLE2_OBJECT,
```

```

SHEET          TYPE OLE2_OBJECT,
CELLS          TYPE OLE2_OBJECT.

```

```
CREATE OBJECT APPLICATION 'Excel.application'.
```

```
SET PROPERTY OF APPLICATION 'Visible' = 1.
```

* 宣告一 EXCEL 工作底稿档(WORKBOOK)

```
CALL METHOD OF APPLICATION 'Workbooks' = WORKBOOK.
```

* 增加一新的工作表(SHEET), 编号是 1 号

```
CALL METHOD OF WORKBOOK 'Add'.
```

```
CALL METHOD OF APPLICATION 'Worksheets' = SHEET
      EXPORTING #1 = 1.
```

* 设定此工作表开启使用

```
CALL METHOD OF SHEET 'Activate'.
```

```
PERFORM FILL_SHEET.
```

```
FREE APPLICATION.
```

*-----

```
FORM FILL_SHEET.
```

```
DATA: ROW_MAX TYPE I VALUE 256,
```

```
      ROWS      TYPE I VALUE 1.
```

```
      INDEX     TYPE I.
```

```
FIELD-SYMBOLS: <NAME>.
```

```
SELECT * FROM KNA1.
```

```
  ROWS = ROWS + 1. "至第 ROWS 列
```

```
  INDEX = ROW_MAX * ( ROWS - 1 ) + 1.
```

```
  DO 10 TIMES. "如要每一列放入 10 个存格的数据
```

```
    ASSIGN COMPONENT SY-INDEX OF STRUCTURE KNA1
      TO <NAME>.
```

```
    CALL METHOD OF SHEET 'Cells' = CELLS
```

```
      EXPORTING #1 = INDEX.
```

```
    SET PROPERTY OF CELLS 'Value' = <NAME>.
```

```
    ADD 1 TO INDEX.
```

ENDDO.
ENDSELECT.