Keywords

CCL保留了一部份在 C 中有相同功能的關鍵字：

break	case	char	continue	default	do
double	else	float	for	if	int
long	return	static	switch	void	while

另外，也有一部分CCL特有的關鍵字，這些關鍵字大多用來詳細定義參數。

Keyword	Description
cmd	相當於C中的"void"
env	這個命令只在特殊環境下有效
hilim	設置參數的上限
list	參數被賦予固定序列中的值
lolim	設置參數下限值
noquery	在系統不作提示下，賦予參數默認值
Onprompt	提示是否賦予參數默認值
Optional	可選擇性的對參數進行詳細定義(非必須)
prompt	在命令行中需要輸入參數時，作具體使用提示
real	同C中的 "double"
setmodule	後續的命令在特定的模塊中說明
ulist	參數被賦予從序列中或鍵盤輸入值

Comments

命令可以用/* ... */,或者在命令行前輸入//來進行界定。CCL中沒有規定指針，結構體，CASTS,這些都不是語言的組成部分。

OSLO Help Overview - Programming - CCL Programming

CCL Procedures

一個CCL的程序具有如下的通用格式：

```
數據類型 Procedure_Name (Argument list)
{
  procedure statement(s)
}
```

參數列表必須具備數據項標識符，並且具有一個或多個入口。

```
Datatype   Argument_Name
```

如果定義的參數多於一個，則參數列表的入口必須用逗號分開表示。如果程序中沒有參數，參數列表可以整體忽略，用簡單的關鍵字"void"代替。具體的可以參看Integrating CCL commands to OSLO中對於參數的定義的幫助部分。

一個程序被另一個程序調用時，必須進行預先的聲明。一種方式是把整個被調用的程序放在所有要調用到該程序的前面。當然，我們也可以用原型聲明來代替，具體格式如下：

```
Datatype Procedure_Name (Argument list);
```

此時整個程序可以放在源文件的任何一個地方。

在CCL里，一個子程序一旦被定義爲static格式，那麼，在程序運行過程中，這個程序不能爲其他程序所調用，但可以被訪問。我們在程序數據項說明前加上關鍵詞"static"來定義該程序爲靜態。比如:

```
static double detval(matrix[][]).
```

"Command line procedures"是可以直接從OLSO命令行中執行的程序。大多數這樣的程序都由簡單的語句或命令表達式組成。儘管如此，它們也可能由CCL程序中的任意語句，包括"while" 和"for"循環。正式的說來，在CCL程序後加上分號表示該程序的終結，但這個分號也經常在程序命令行後被省略。

在命令行中的任何一個表達式都可以立即計算並把結果顯示在命令行後的信息區。因此命令行在提供命令入口的同時也能做爲計算器使用。

多命令行程序可以在CCL的編輯文本窗口中輸入,編輯框中的任何一部分都已可以被選中並立即執行。但是，UltraEdit沒有這個功能。

OSLO Help Overview - Programming - CCL Programming

CCL Differences from C

如上所述,CCL於C非常的相似。CCL支持所有C的控制語法,除了條件表達式"(a) ? b : c"和"goto"語句。CCL支持C中的大部分操作，並有一部分的數據類型一致(不包括指針，結構體，或聯合體)，但在一些情況下，CCL和C之間也有本質上的區別。文章的這個部分，描述了這兩種語言之間的主要不同之處。

Floating point array indices

CCL允許使用可變浮點點或者表達式作爲數組下標，這個時候，下標會被自動的轉換爲整數值。同時，在CCL中，可以用表達式來定義非靜態局部可變數組的維數。

Global scope of argument names

在CCL中，一旦一個參數被定義了，它的名稱和類型就已經爲整個程序所知。因此，如果某個參數在一個子程序中定義爲"int count"，那麼其他子程序里就不能有

"double count"之類的定義出現。同時，在CCL中，參數名稱不分大小寫。

Static procedures

在CCL中，一個子程序如果被定義爲static時，就表明這個子程序只能被其他程序訪問，但不能被調用。

Optional parentheses in procedure calls

一般的，程序參數後附有括號，用分號分離。在CCL中，請求語句（或void函數），可以把括號省略，甚至也可以把分號省略。既然空格可以分離參數，表達式（比如a+b）作爲參數使用時，必須注意不要在字符之間留下空間。如果參數列表被附以括號，參數間也用分號分離，那麼，作爲參數的表達式中間可以有空格。

如果調用的函數中沒有參數，那麼分號也可以被省略。因此，如果time()是一個函數，那麼"a = time();" 和 "a = time;" 是等價的。

例子：

假設命令的原型聲明如下：

```
cmd plot (real x1, real y1, real x2, real y2);
```

在程序中調用該命令時，可以有如下的表達方式：

```
plot(2.4,3 * x + 4 * sqrt(y),z, z + 1);
plot 2.4, 3*x+4*sqrt(y), z, z+1;
plot 2.4 3*x+4*sqrt(y) z z+1;
```

在程序運行時，一般多採用最後一種（如果在後面沒有另外的命令，則冒號也可以省略，但在C中就不可以）。

Exponentiation

求冪的操作(**)是從FORTRAN借鑑而來。 a** b表示a的b次冪。a和b都可以是整數或者小數。如果a和b都是整數，那麼結果將不大於2147483648，並且是整數型。否則的話，就是雙精度類型。當b=2時，那麼結果就是a自己乘以自己，但如果b!=2，則，就需要調用C函數pow(a, b)才能得到結果。如果a=0而b是負數，結果是會出現"divide by zero"的錯誤提示。如果a是負數而b不是一個整數，結果將出現"exponent error"（指數錯誤）的錯誤提示。

The Null Statement

在C中，一個單獨的(;)被稱之爲空語句，沒有實際意義。但在CCL中，空語句有可能被定義爲OSLO的一個固定命令，並在程序運行時執行定義的操作。在程序運行時，在空命令行處按下回車鍵，會被當作空語句執行。在OSLO的鏡頭編輯器中，上

述的動作將導致提前進入後續透鏡的表面。

The "Pass by reference" & Operator

在C和CCL中，通常使用標量參數的副本和數組參數的地址去訪問子程序。因此，若被訪問的程序里可變標量參數的改變並不會影響到訪問者。有時候會有必要把標量參數的地址傳遞給被訪者，這樣，就可以重新設置訪問者中該變量的值了。在C中，被訪問的程序中會將這樣的參數定義爲指針形式。CCL中沒有指針概念，但是在程序中如果我們在可變標量參數的名字前加上"&"，被訪問程序就能掃描到該參數的地址了。比如，變量除了通過值來傳遞之外，還可以通過標誌符傳遞，訪問函數不知道該變量是通過值還是標誌符傳遞。

將可變標量以值或地址的方式傳遞，類似於C++的某個特點。這也彌補了CCL沒有指針的不足。

Omitted Arguments

一旦一個子程序被調用，無論這個程序從命令行或者從另一個程序而來，標量參數，或者一維字符數組都可能被忽略。如果這種情況出現，系統會給用戶以輸入提示，或者直接賦予其默認值(跟據參數相關的部分推得)。如果是整數列或者實數列被忽略，那麼編輯器將會發出錯誤訊息。

具體可以參考Integrating CCL commands to OSLO。

The Question Mark argument

當一個程序被調用時，參數部分的問號(?)提示用戶輸入必要的參數。有兩種情況：一黨問號在參數中間時，如plot 2.4 ? z z+1，那麼，只需要用戶輸入第二個參數；二如果問號緊隨著參數列表，如plot ?，則表明可以輸入任意個參數。

具體參見 Integrating CCL commands to OSLO 關於參數定義部分。

Integer Division

在C中，除以一個整數的結果肯定也是一個整數。比如，22/7＝３。但是在CCL中，將會得到真實的結果，如22/7＝3.142857。但是，如果將得到的結果賦予一個整數變量，那麼，得到的結果也將是一個整數，如定義j爲整數，若j=22/7，得到結果是j=3。＋

Case-switch 語句

Case-switch 中的控制表達式，通常被限制在從-32767 到 32767 範圍內的整數常量。在 C 的程序中，#define 聲明部分可以控制表達式進行命名，如下所示：

```
#define ALPHA 1
#define BETA 3
switch (value) {
/* Note: CCL provides a library command "printf" that is equivalent to the
```

```
C library printf
 case ALPHA:
 printf("Alpha selected\n");
 break;
 case BETA:
 printf("Beta selected\n");
 break;
 default:
 printf("Nothing selected\n");
 break;
 }
}
```

如果在"switch"表達語句中,估計量做爲 double 類型,那麼將自動取其整數部分。

二維數組：

CCL 和 C 在對二維數組自變量 array[][]的說明上有很大的不同。在 C 中，必須給定列數，比如: array[][20]。而在 CCL 中，數組維數被"秘密的"傳遞到程序中了,因此，程序會把它作爲一個行列數都可變的數組處理。CCL 同樣提供了庫函數(rowdim,coldim)，通過相關的庫函數，可以獲取以變量形式出現的二維數組的行列數。數組維數的有效性不僅僅只限制於用戶定義的數組，並且也限制了 OSLO 定義的數組；同時，在 CCL 中可以使用表達式而非常量，來定義局部數組變量的維數。以上的兩個特點大大便利了對於矩陣的操作。

例如:

(1)以字符串(數組)和字符數作爲自變量的命令。

```
Command prcstr(char string [] ,int count)
{
procedure statement(s)
}
```

(2)一個可實現對二維浮點數組的操作,並返回值是一個浮點數值(比如矩陣中具有決定性作用的值)的函數

```
double detval(double matrix[][])
{
procedure statement(s)
}
```

邏輯表達式

在 C 中,邏輯表達式被從左到右的估計,直到表達式最後狀態的確定。在一些情況下,不必要估計整個表達式來決定它的最終狀態，而在 CCL 中,則常常需要估計整個表達式。

ScanCCL Searching Utility

掃描 CCL 文件中的函數和變量名。通過 Scanccl 可以找到已經存在的 CCL 文件的信息，無論這個文件在公共的目錄，用戶的目錄。返回的信息是：

函數：名稱,數組,返回類型,定義在的文件和行的位置

數組：名稱,類型,被使用的次數,所在的文件和行數

全局變量：名稱,類型,所在文件的文件名及所在行

每個項目(函數,數組,全局變量)都可以用名稱，或文件名來分類。

CCL 編譯器

CCL 命令

缺省菜單：Tool > compile ccl

主工具欄中圖標爲

注意：只有在 OSLO 缺省文本編輯器中編譯才是自動的。如果使用的是 ULtraedit ，則需要進行編譯操作。

當文件以後綴.ccl 存儲時，保存的同時，該文件會被自動編譯，並可被立即執行。當然，若在編譯過程中發現有錯誤，系統將會在當前文本窗口中顯示錯誤，以方便改正。

若不想讓文件被自動編譯，可以將文件以.ccx 後綴命名，保存在 CCL 子目錄中。如果一個.ccl 文件有錯誤,就無法使用 CCL，所以使用.ccx 後綴在處理中標志。通過設置 ccac 爲 OFF 也可以阻止防止自動編譯。

CCL 編譯器在公共目錄還是在用戶目錄下運行，主要取決於被編譯文件被保存在哪個目錄下。然而，若公共目錄下的 CCL 程序被重新編譯了，則所有依附於該 CCL 程序的用戶目錄下的 CCL 程序都將被重新編譯。在一個多用戶系統中,當其他用戶正使用程序時，不要去重寫公用的 CCL，因爲那樣會使他們的 CCL 文件不可再用。如果公用的 CCL 已經被重寫，則每個用戶，從第一個啓動 OSLO 的人開始，都將會收到通知，並提示重新編譯該用戶 CCL。

將 CCL 程序分爲公共目錄和用戶目錄保存，可以將大型的用戶 CCL 程序中的大多數 CCL 程序移到公共目錄下，這樣，就可以防止由於某個小地方的修改而導致用戶目錄下整個大程序被重新編譯。

雖然通常狀況下 CCL 編譯器自動執行，但也可以通過 ccl 命令控制其運行。變量爲 public,private,后跟文件類型：command,manu,list,string,all。當 CCL 程序通過外部編輯器實現功能擴展時，此種編譯模式是很有用的。

Ccl 編譯命令 compile 中第四個變量 char ccl_opts[]是可選的，且系統不會對該變量進行提示。若該命令被輸入到命令行，.其將被附加於執行的命令字符串后。該字符串變量可被用來指定特定的編譯選項(e.g size size size) 。

Example: **ccl public all " 500000 200000 "**

Ccl_opts 字符串中可能被指定的可選項有如下几種，這裡的 size 都表示一個整

數所佔的字節數。

- g size: 指定分配給存儲 ccl 全局變量和常數用的總空間。CCL 編譯器在啓動時就分配好了這個空間。不過，OSLO 中需要的空間並不需要被指定，而是根據在 ccl 編譯過程中所需要的實際空間而定。最大的字節數爲 8388608.。該默認值爲 1048576。
- l size: 指定 OSLO 中用來暫時存儲局部變量和標量的堆棧所佔用的空間。當堆棧從 CCL 命令或函數中返回時，該堆棧中的內容原有的內容將被清空，堆棧可重新利用。堆棧最大的佔用字節數可爲 8388608。默認值爲 262144。
- c size: 指定分配到 OSLO 中的地址空間，這些空間，用來存儲在內存中的已經被編譯的 ccl 代碼使其可被重新利用。這個空間至少要和最長的那個 CCL 程序一樣大。當已編譯好的程序超過該存儲地址的空間大小時，此 CCL 編譯器將失效。該空間越大，則將允許越多的 CCL 代碼進入內存，從而可以較少的硬盤中讀取代碼。此空間的最大值爲 8388608。默認值爲 131072。
- -D name [=value] 定義常數的名稱及其值。這個命令和 CCL 代碼中的 #define 等效。

OSLO Help Overview - Programming - CCL Programming

Setmodule

在構建一個程序時，我們經常會用相同的命令，但根據不同的環境，選擇不同的參數，使之發揮不同的作用。比如說，在光學設計過程中，會需要用一個簡單的關鍵字，進行曲率半徑的控制和表示，而不考慮設計的是鏡頭入口，結構入口還是對全局的編輯。AppMan通過使用"modules"(模塊)來處理這樣的情況。特別的，在OSLO中有一個“L”模塊包含了鏡頭的入口命令，有一個“C”模塊包含了結構入口命令。因此，繼續以上的例子，在OSLO里有三個“rd()”命令:一個在L模塊中，一個在C模塊中,還有一個不在任何的模塊。在特定的模塊中，“rd()”命令將執行不同的操作。如果沒有被明確的聲明在哪個模塊中，命令默認爲是針對程序全局。

在CCL中，關鍵字“setmodule”用來表示模塊的不同應用。如果我們需要在程序中用到一個特殊的模塊，那麼，必須聲明“setmodule L;”或“setmodule C;”作爲函數的一部分。Setmodule將通知編譯器使用哪個命令。其應用如以下的例子：

```
{
  configuration_edit(update);
  setmodule C;
  rd(ins, 3, 2, 25.0);
  end();
}
```

OSLO Help Overview - Programming - CCL Programming

Error Handling

一般的，當CCL或SCP的內部命令在運行中出現錯誤時，會在程序的最後出現錯誤訊息。通過安裝慣用的程序來處理由這些內部命令產生的錯誤。具體實現用如下命令：

```
install_error_handler (char Error handler name[])
```

錯誤處理器（error handler），使CCL或SCP在遇到內部命令產生的錯誤時，仍能通過採取特殊的措施，讓程序繼續運行。

當程序內部命令返回錯誤代碼時，install_error_handler將對這命令程序進行指定命名。錯誤處理器沒有參數也沒有返回值，它是一個命令，而不是一個函數。如果錯誤處理器是一個SCP程序，那麼特定的錯誤處理器的名字的第一個字母必須是星號（*）。

一旦錯誤處理器被訪問，CCL的全局整數變量:Errno，將被設置爲由內部命令返回的錯誤代碼。在錯誤處理器中執行"printf(Errno)"，或者由" message(Errno)"發出錯誤訊息。 錯誤處理器設置初始值Errno = 0，若Errno爲0，則當前命令將繼續執行，產生的錯誤將被忽略。若錯誤處理器有了返回值，則命令會被立即終止。

錯誤處理器一旦爲install_error_handler命令指定，則會一直使用，且只能用如下命令使其停止:

```
delete_error_handler(void)
```

黨delete_error_handler命令被執行，在出現錯誤後，錯誤處理器會先發出錯誤訊息，然後將程序自行停止。

一般的，錯誤處理器必須在返回之前從安裝程序中刪除掉，否則錯誤處理器將普遍代替程序中的默認行爲。當前錯誤處理器的名字存儲在全局可變error_handler中。

以下是一個在SCP中關與錯誤處理的簡單程序：

```
*errhand
printf("Entering error handler...\n");
printf(Errno);
printf("Continuing...\n");
Errno = 0;
```

默認的錯誤處理命令是"prefs"，這個命令可以保證一些關鍵參數的返回：

output_text, graphics_autoclear, graphics_axes, graphics_labels。

如果你需要使用另一個錯誤處理命令，那麼，在程序的結束時要保證恢復爲默認的"prefs"命令。

OSLO Help Overview - Programming - CCL Programming

Integrating CCL to OSLO

OSLO的設計完全用戶化。這包含了：

- 功能強大的編程語言： CCL
- CCL程序的"OSLO-like"用戶界面。An "OSLO-like" user-interface to CCL routines

下面將對第二點作詳細敘述。

Making the most of CCL: arguments

如前所述，在CCL程序中,CCL程序和命令可以有變量參數。

定義一個參數需要兩個條件:

- 參數數據類型:
  整數類型用int表示(int[]表示數組，int[][]表示矩陣)
  浮點數用real或double表示(real[]表示數組，real[][]表示矩陣)
  字符用char表示(char[]表示一個字符串)
- 參數名

以上這些都在("my_ccl.ccl")文件中有自己的功能函數。

```
int square(real x)
{
return x*x;
}
```

另外CCL中還定義了一些附加的參數範圍：

- a lower limit: lolim
  系統能接受的最小值
- a higher limit: hilim
  系統能接受的最大值
- a prompt string: prompt
  對輸入部分標示。一般默認標識符爲該變量參數名。
- a list of items to chose from: list
  在(PUBLIC or PRIVATE)/CCL/INC/A_LIST.H中定義的列表。列表中的參數只能定義爲整數形式。具體使用如下：
  - 二選一：
    list output

```
{
"text" = 0,
"graphic" = 1,
}
```

選項也可以用短句形式定義 ("a_list.h"):

```
list output_graphic
{
    "{con}contour plot" = 0,
    "{map}color map plot" = 1,
}
```

- 建立動態輸入：一個函數中的參數會因爲其前面參數的值而被忽略。比如：

```
"a_list.h":
list list_output
{
"text" = []0,
"graphic" = [1]1,
}
list list_graphic_output
{
"{con}contour plot" = 0,
"{map}color map plot" = 1,
}
"my_ccl.ccl":
int output {list="list_output"}
int graphic_output {list="list_graphic_output"}
cmd wavefront(int output, int graphic_output)
{
...
 }
```

黨執行wavefront命令時，OSLO會提示用戶選擇text或graphic輸出。如果用戶選擇graphic，OSLO會緊接著提示用戶輸入graphic_output值。這是因爲在程序前面有"graphic" = [1]1，方框欄中的數字標示建議的參數個數（從0開始計數）。如果用戶選擇的是text，那麼接下去的graphic_output將被忽略。因爲"text"=[]0，這就表示沒有其他的參數值需要輸入了。

- a default value: