32 High-Level Language Module (hll.hhf)

The hll.hhf library module adds a switch/case/default/endswitch statement that is similiar to the Pascal case statement and the C/C++ switch statement.

32.1 The HLL Module

To use the high-level language functions in your application, you will need to include one of the following statements at the beginning of your HLA application:

```
#include( "hll.hhf" )
```

or

```
#include( "stdlib.hhf" )
```

32.2 The switch/case/default/endswitch Macro

```
#macro switch( reg32 );
#keyword case( const_list );
#keyword default
#terminator endswitch
```

A commonly used high level language statement missing from HLA's basic set is the the C/C++ switch statement (the case statement in most other languages). The SWITCH/CASE/DEFAULT/ENDSWITCH macro set in the hll.hhf header file provides this missing HLL statement.

The HLL module's switch statement actually provides two different user-selectable syntaxes. The first is a Pascal-like syntax. It takes the following form:

```
switch( reg32 )

  case( constant_list )
     <<body>>

  << additional, optional cases >>

  default       // This section is optional too!
     << body >>

endswitch;
```

As you might expect, the reg_{32} parameter has to be an 80x86 32-bit general purpose register. The *constant_list* operand has to be a sequence of one or more positive ordinal constants. There must be at least one *case* present in the statement (default does not count as a case) and there may be a maximum of 1,024 cases in the *switch* statement. Furthermore, the range between the largest and smallest values for all the cases must be less than or equal to 1,024. Note that, unlike C/C++, you do not end each case with a *break* statement; nor does control fall through from one case to the next. Here is a simple example of a Pascal-like *switch* statement:

```
switch( ebx )

  case( 1 )
     stdout.put( "case 1 encountered" nl );

  case( 3 )
     stdout.put( "case 3 encountered" nl );

  case( 10 )
     stdout.put( "case 10 encountered" nl );
```

```
    case( 15, 20, 25 )
       stdout.put( "case 15, 20, or 25 encountered" nl );

    default
       stdout.put( "Some other case was encountered" nl );

  endswitch;
```

Although C/C++ semantics for a switch statement are stylistically inferior to Pascal, some people might prefer a C/C++ version of the switch statement. The HLL switch statement uses a special predefined boolean VAL constant, *hll.cswitch*, that lets you choose C/C++ semantics. By default, the *hll.cswitch* constant is set to false. By placing the statement "?hll.cswitch:=true;" before a *switch* statement, you can instruct the *switch* macro to use C/C++ semantics rather than Pascal semantics. The difference between the two is that for C/C++ semantics you must end each case with an explicit *break* statement. The Pascal version is preferable since it is slightly more efficient and a bit more readable.

By default, the *switch* macro uses a quicksort algorithm built into HLA's @sort compile-time function to sort the cases when building the jump table that the *switch* statement compiles into. For the vast majority of *switch* statements you'll write, this is a good choice. However, if you create a really large *switch* statement and the cases you supply are already sorted in ascending order (or mostly sorted), a bubble sort will actually outperform the quick sort algorithm. In this (very) special case, you an improve the compilation (not run-time) performance of the *switch* macro by adding the following statement immediately after the switch statement:

```
  switch( eax )
    ?hll.usebubblesort := true;

    <lots of pre-sorted cases>

  endswitch;
```

Note that this trick speeds up compilation only if the cases are already sorted in ascending order. If they are not sorted in ascending order, then the bubblesort algorithm is *much* slower than the quicksort algorithm and you shouldn't use it.