## 20. SEPARATE COMPILATION *)

According to the Common Base definition, separate compilation of procedures and classes may be introduced in a Common Base implementation.

When a program using a separately compiled class is compiled, all identifiers local to the class must be known by the compiler. This may be solved by making a separately compiled class consisting of two parts: a name table (with type indication and relative address) and the object code.

It is permitted to restrict the use of a separately, compiled class (and also a system class) in the program by enforcing the rule that an external declaration of a class C may only appear on one single block level within the program.

This does not prohibit the use of an external declaration for the same class in two blocks with disjoint scope if these are on the same static level.

## 21. Store_collapse

### 21.0 Main_flow_of_store_collapse

The store collapse consists of six phases:

1. Start on CD, locate all referenceable objects.
   MDP of referenceable objects will at the end of this phase point to itself.

2. Scan POOL 1 sequentially.
   Compute addresses of objects in POOL 1 after move (in phase 5) and store in MDP.
   Chain first block of each available area to the next used block to speed up later phases.

*) The problems of Separate Compilation are currently being studied by the SIMULA Standards Group.

3. Move POOL 2 by minimal moves.
   Update pointers from POOL 2 to POOL 1.

4. Scan POOL 1 sequentially.
   Update all pointers in POOL 1.

5. Move POOL 1. Zero MDP.

6. Scan POOL 2 sequentially.
   Update POOL 2 pointers to POOL 2.
   Insert new MDPs.

## 21.1 Phase_1

During phase 1, an object chain is used to save objects in which pointers have not yet been followed. This object chain is, first-in last-out. Each object, using MDP, points to its predecessor. The last object is pointed to by a ref variable object declared in the runtime system.

Whenever a reference to an object is found, the procedure chain is used. This procedure will put the object on object chain, provided:

1. It is not in the object chain.

2. It has not been fully processed before.

This may be determined by MDP of the process, which during phase 1 may be in one of three states, as follows:

1. MDP points to the object itself:

   The master driver (if any) of this object has been marked as referenced and all pointers in the object have been followed.

2. MDP points to another object or MDP = none and objch or another object points to this object:

   The master driver (if any) of this object has been processed, but the pointers in the object itself have not yet been followed.

3. MDP points to a driver or MDP = none and neither objch nor another object points to this object:

   It has not yet been found that this object is referenceable.

When all pointers in an object have been processed, the next object is removed from the object chain, its MDP set to point to itself and then all pointers local to this object are processed.

### 21.1.1 Asgn

Asgn is a utility procedure used by the procedures first-pointer and nextpointer to set up the non-local variables kin, typ, pa and va declared local to storecollapse, and to increment the pointer index pnr.

### 21.1.2 Firstpointer, nextpointer

Firstpointer and nextpointer are procedures which locate pointers in an object.

A call on the procedure firstpointer has two parameters:

1. A ref to the object where pointers should be located.

2. A label to which control will be transferred when no more pointers are found in the object.

   If there are no pointers, the procedure will at once go to the label parameter.

If there is at least one pointer, firstpointer will return with the address of the first pointer as function value. It will also note the parameters for future use in the variables pobj and exit declared local to storecollapse.

Each successive call on nextpointer will then return as function value the address of the next pointer.

The index of the current pointer is recorded in the integer pnr local to storecollapse.

When no more pointers are left in the object, nextpointer will go to the label given to firstpointer as a parameter (now found in exit local to storecollapse).

### 21.1.3 Chain

Chain is used to insert objects on the object chain if they are not already on this chain or have been previously processed.

### 21.1.4 Map

The procedure map will put objects on the object chain and mark their drivers as referenced. In some cases, drivers will be inserted on the driver chain.

Map may be used to map an entire dynamic structure, starting in the innermost quasi-parallel system of the structure, or to map a static structure, such as the drivers and objects for terminated objects.

A step by step description of map is given below:

Step 1:

If the driver has been processed before, exit.

Step 2:

If drex of this driver is none, we are going to process a static structure. In this case continue from step 13.

Step 3:

Locate the nearest detached object or prefixed block.

Step 4:

Locate operating object of innermost quasi-parallel system.

Step 5:

Mark this driver as referenced.

Step 6:

If this is a connector driver and the extra static link (cdrp) is none, this driver is included in the driver chain for later processing of cdrp. Continue on step 9.

Step 7:

If dot is true, i.e. the driver has been created when a procedure was called by remote referencing, the driver pointed to by the static link (drp) is put on the driver chain for later processing. Continue on step 9.

Step 8:

Put the accumulator stack on the object chain for later processing.

Step 9:

Put the object on the object chain for later processing.

Step 10:

If this was not the driver of a detached object or a prefixed block, follow the dynamic link (drex) once and proceed from step 5.

Step 11:

If this was the driver of a prefixed block, follow the static link once and proceed from step 5. (A prefixed block has no dynamic link.)

Step 12:

Follow static link once and proceed from step 1. (A detached object has no dynamic link).

Step 13:

Mark this driver as referenced and put it on the driver chain for later processing. Follow static links once and proceed from step 1.

21.1.5 Maptree

The procedure maptree is used to mark all eventnotices and drivers of processes they refer to as referenced, and put processes on object chain when necessary.

The algorithm supposes that the sequencing set is organized as a binary tree (cfr. section 20.3). Furthermore, it assumes that a left branch which is empty implies that the right branch for this node is also empty.

There exist several possible algorithms. The method chosen here requires a fixed amount of working storage to handle sequencing sets of any size.

The algorithm starts from the uppermost node of the tree. This is pointed to by the ref (EVENTNOTICE) variable high in class SIMULATION. At this point, all EVENTNOTICES in this sequencing set have referenced false:

1. Proceed to the right until an EVENTNOTICE is found which has either no right brancn or is previously marked as referenced.

2. Proceed to the left until an EVENTNOTICE is found which has no left branch.

3. Put the process referenced by this EVENTNOTICE on the object chain (using chain), and mark this EVENTNOTICE as referenced.

4. Follow the backward link from this EVENTNOTICE.

5. Put the process referenced by this EVENTNOTICE on the object chain (using chain) and mark this EVENTNOTICE as referenced. All EVENTNOTICES to the right of this event-notice must have been marked previously.

6. Follow the backward link from this EVENTNOTICE, if there is none, the whole sequencing set has been marked, if not proceed from 1.

It is possible to use a similar algorithm to the one described here starting on low.

### 21.1.6 Chain_2

Chain 2 is used to insert drivers on the driver chain drchn for later processing.

## 21.2 Phase_2

Phase 2 is a sequential scan of POOL 1.

During this scan:

1. Address after move of each referenceable object is computed and stored in MDP of the object for fast lookup when updating pointers in phase 3 and phase 4.

2. Adjacent available blocks are combined into one block by making PP none and MDP a pointer to the next referenceable block.

21.3 Phase 3

Phase 3 is a minimal move of POOL 2 and update of all pointers from POOL 2 to POOL 1.

When the contents of a notice is moved, an indication where it has been moved is put into the old notice for updating purposes.

The algorithm is as follows:

1. Start from the bottom of POOL 2.

2. Locate a notice which is not referenced.
Update POOL 1 pointers in all referenced notices found during this scan. If we have come to the top of POOL 2, we are finished.

3. Start on top of POOL 2 and decrement the size of POOL 2 until a referenced notice is found.
If we during this reach the notice found in 2, we are finished.

4. Move contents of notice found in 2 to available space found in 3.
Update pointers to POOL 1.
Record where the notice has been moved.

5. Decrement POOL2 size by one notice and increment the other pointer by one notice.
If the pointer is not outside POOL 2, proceed from 2.

6. Perform some housekeeping to figure out the two separate cases when the pointers meet.

At the end of phase 3, it is poosible to compute the amount of available storage and check that this covers the amount required. This is not done in the algorithm shown here. The test has been postponed to after phase 6.

### 21.3.1 Move

The procedure move has three arguments:

1. Address to move from, x.

2. Address to move to, y.

3. Length of area to be moved, i.

The continuous area of storage (x,x+i-1) is moved to (y,y+i-1).

### 21.3.2 Updl

updl will update pointers from POOL 2 to POOL 1:

1. Pointer to the object, obj.

2. If notice is a driver, pointer to the accumulator stack, acs.

## 21.4 Phase 4

Phase 4 is the updating phase for POOL 1. It is a sequential scan of POOL 1, using the shortcuts established by phase 2 combining adjacent available areas.

All pointers are updated. For POOL 1 pointers, the new value is found in MDP of the referenced object.

For POOL 2 pointers, the new value is recorded in the old notice if it has been moved. Only POOL 2 pointers pointing to the area between POOL 2 top when the store-collapse was entered and POOL 2 top after phase 3 should be updated; as only these have a new address after move.

Note:

Since the outermost block of a program must be resident from the point of view of the runtime system, the area starting at POOL 1 first is always referenceable.

## 21.5 Phase 5

Phase 5 is a move of objects of POOL 1.

POOL 1 is scanned sequentially using mfa.

mta contains the address where the next referenceable object should be moved.

MDP is set to none.

The actual move is performed only if mfa is not equal to mta.

A continuous referenceable area is moved by one call on the move procedure (cfr. section 23.4.1).

## 21.6 Phase 6

Phase 6 is a sequential scan of POOL 2 from starting at POOL 2 bottom and some housekeeping to prepare for exit from the store collapse.

Pointers to POOL 2 in the notices are updated using upd2.

CD is updated.

The available storage list is cleared.

If the required storage exceeds available storage, an error condition is raised.

21.6.1 Upd2

upd2 will update a pointer to POOL 2.

If the value of the pointer is less than the POOL2TOP determined in phase 3 (i.e. outside POOL 2), the new pointer value will replace the old value.