Memory Allocation

Introduction

Normally all memory allocation / deallocation is done by Ada (The Ada Runtime Environment) without the programmer having to think about it. Whenever a new variable is created memory is allocated for it and if needed the programs wimpslot extended. The user only has to make sure the wimpslot is big enough initially at startup after that the program takes care of itself.

Wimpslot of a program.

There are however some cases where a more direct means of allocation and deallocation is needed which is explained in this chapter.

Normal allocation

The basic problem is that you want a block of memory and you want control over when its allocated and de-allocated. The obvious and easiest way of achieving that is using a '`declare`' structure to allocate a variable.

```
declare
   Block : String(1..1024);
begin
   -- Block is allocated and can be used.
end;
-- Block is deallocated.
```

This is still quite inflexible and therefore RASCAL has procedures to allocate and deallocate memory blocks directly.

```
with Memory;
procedure Example is
   Block : Memory.mem_adr_type;
begin
   Block := Memory.Allocate (1024);
   -- Block is allocated and can be used.
   Memory.Deallocate (Block);
   -- Block is deallocated.
end Example;
```

The above should be all you need in most cases, but if your program is going to do a lot of allocations or deallocations or its handling very big blocks of memory then you should consider a different means of allocating memory. The reason being that while the wimpslot is extended automatically when the need arises it is never reduced again. Deallocating will free the memory for further use by the program but it will not reduce the wimpslot and the memory is therefore not available for other programs even though it is not used by your program.

This is not a problem as long as you program is handling ’normal-sized’ data but if you are handling pictures or anything with sizes like that you should consider using the allocation method described in the next section.

Flex heap allocation

The flex heap is a library created to solve the problems mentioned above. To use it you need to link it with you program by adding "`-largs RASCALlib:flexlib-32.o RASCALlib:flexlibzm.o`" to the compile statement resulting in something like this:

```
gnatmake RunImage -cargs -mapcs-32 -LRASCALlib: -IRASCALlib:
-largs -mstubs RASCALlib:flexlib-32.o RASCALlib:flexlibzm.o
```

The ’Flex’ package in RASCAL is a thin binding to this library and the ’Heap’ package is a thick binding (which uses the ’Flex’ package). It is recommended that you use the Heap package and the following will be solely about using the Heap package.

The flex library create a new memory area at end of the Wimpslot. This area can both grow and shrink and it will defragment to keep memory requirements to a minimum.

Wimpslot of a program with flex heap.

Using the flex heap does not prevent the use of normal allocation methods. If the normal allocation area needs to be expanded the Wimpslot is extended and the flex area moved to give space.
All this flexibility means that the memory blocks allocated in the flex area do not stay at the same place - the address of such blocks may change. The Heap package associates every flex block with a variable (heap_block_type) and this is always kept uptodate with regard to the address of the block. Therefore you should always ask this variable for the address of the block prior to accessing the memory block.

```
declare
   Block : Heap.Heap_block_Type(1024);
   -- Block is allocated and can be used.
begin
   -- Flex heap blocks can be extended.
   Heap.Extend (The => Block, New_Size => 2048);
   -- Always get the uptodate address.
   Memory.StringToMemory (Str => "Example",
                          Adr => Heap.Get_Address (Block));
end;
-- Block is deallocated.
```

The flex heap can also be created as a dynamic area. To achieve this you must define the max. size of this dynamic area before the first flex block is allocated. The default size is zero and means that no dynamic area is created and the wimpslot used instead. You can also

define the name of the dynamic area, the default name is "RASCAL -Heap".

```
procedure Example is
begin
   Heap.Set_MaxSize (32*1024*1024);
   Heap.Set_Name ("Example's Heap Area");
   declare
      Block : Heap.Heap_block_Type(1024);
      -- Block is allocated and can be used.
   begin
      -- Flex heap blocks can be extended.
   end;
   -- Block is deallocated.
end Example;
```

Wimpslot of a program with flex heap in a separate dynamic area.