

Real-Time Programming

Summer Semester 08

Assignment #1

Homepage: http://www.iste.uni-stuttgart.de/ps/Lehre/SS2008/V_RTP/

Discussion of Solutions: April 23, 2008

1 Estimation of Memory Requirements

The worst-case memory requirements of an executing program is important to know for embedded systems in order to assure that programs cannot run out of memory during execution.

1. Provide a worst-case estimate for the memory requirements incurred by the heap sort implementation given at

http://www.iste.uni-stuttgart.de/ps/Lehre/SS2008/V_RTP/heapsort.adb

assuming that heap sort is called as follows:

```
Heap : Heap_Type := (1,2,3,4,5,6,7,8,9,0);  
HeapSort (Heap);
```

For your estimation, assume that the compiler has given you the following statistics about the size of the activation records (ARs) of subprograms:

Subprogram	Size of AR in bytes
Main	116
Swap	36
Is_Root	24
Parent_Index	24
Left_Child_Index	24
Left_Child_Exists	28
Right_Child_Index	24
Right_Child_Exists	28
Dump	28
UpHeap	28
DownHeap	40
Remove	32
HeapSort	32

2. Provide a worst-case estimate for the memory requirements incurred by the quick-sort implementation given at

http://www.iste.uni-stuttgart.de/ps/Lehre/SS2008/V_RTP/quicksort.adb

assuming that the following array is to be sorted:

```
A : Array_Type(1..100);  
Quick_Sort_A (A);
```

For your estimation, assume that the compiler has given you the following statistics about the size of the activation records of subprograms:

Subprogram	Size of AR in bytes
QuickSort	428
Swap	36
Dump	28
QuickSort_A	52

Hint: You may need to consult the literature to find out about worst-case call stack depth of quicksort to be able to answer this question.

3. Sketch a strategy to determine worst-case time requirements for quicksort by means of actual testing.
Again, the hint: You may need to consult the literature about quicksort.
4. Name programming constructs that make it hard (or impossible) to estimate the memory requirements statically.

2 Heap Management

Consider the following program

```
for i in 1..1_000_000 loop  
  A := new T7; B := new T9; C := new T3; D := new T5;  
  ... E := D; ...  
  Free(A); Free(B); Free(C); Free(D);  
  -- Free(E); -- (1)  
end loop;
```

The heap is organised as a freelist with “first-fit” strategy. The size of an allocation is given by the type name, e.g. an object of type T7 needs 7 words and of type T3 3 words and so on.

1. Draw the freelist after the first iteration of the loop.
2. Describe the behaviour of heap management of further loop iterations. Does this program work? Explain your answer.
3. Let the commented line at (1) be uncommented from now on. Describe the effect of this `Free` call and the behaviour of further loop iterations.