Pseudo-code
-draft-

M. Pellegrini*

February 28, 2002

1 Introduction

Several books an algorithms use pseudo-code to describe algorithms. This is a mixture of structured control (loops, if-then-else), basic types and aggregates (array, records), assignments, arithmetic operations and parameter passing conventions.

The main difference between pseudo-code and code is that (1) occasionally we use plain English to explain part of the code that is thus not fully specified for a machine (although should be clear to an educated human); (2) type declarations are avoided in pseudo-code since these are necessary mainly for static type checking; (3) the allowable aggregates and basic types are not limited (for example, we might have sets, although few programming languages provide sets as primitive objects); (4) in pseudo-code we use standard mathematical notation (e.g. "=" for equality test, instead of "==" as in C). In pseudo-code we tend to avoid issues like: I/O, buffer management, memory management, round-off errors in floating point arithmetic, error handling, interrupts, human computer interface, etc...

Pseudo-code is usually *imperative*, since in this case the mapping to RAM machine level code is straightforward. Historically, *functional* languages, like LISP, or *logical* languages, like Prolog, tend not to have straightforward implementations in RAM thus making complexity analysis of such algorithms a rather daunting task.

2 Thomas H. Cormen, Charles Eric Leiserson and Ronald L. Rivest

Thomas H. Cormen and Charles Eric Leiserson and Ronald L. Rivest [CLR90, pages 4-5] describe the following pseudo-code:

*Istituto di Matematica Computazionale del CNR, pellegrini@imc.pi.cnr.it

1. Indentation indicates block structure.

2. *While, For, Repeat* loops and *if-then-else* conditionals have the same interpretation as in Pascale.

3. Symbol "▷" denotes comments.

4. Multiple assignments are read right to left.

5. Variables have scope local the the procedure.

6. Array elements are accessed by "name[position]"; sub-arrays are accessed by "name[begin..end]"; array begin default with index 1.

7. Compounds are organized as objects with attributes; the attribute of an object is accesses with "attribute[object]"; A variable representing and object is treated as a pointer to the object.

8. Parameters are passed to procedure by "value". Side effects into the caller environment are realized by object/attribute assignments.

Example Quicksort in C:

```
int partition(int *inarray, int init, int end);

void quicksort(int *inarray, int init, int end)
{int q;

if (init < end)
    {
    q = partition(inarray, init, end);
    quicksort(inarray, init, q);
    quicksort(inarray, q+1, end);
    };

}

int partition(int *inarray, int init, int end)
{
int pivot, i,j, tmp;
pivot = *(inarray+init);
i=init-1;
j=end+1;
while (1==1)
    {
    do {j--;} while  ( *(inarray+j) > pivot );
    do {i++;} while (  *(inarray+i) < pivot ) ;

    if (i < j)
        {//swap
        tmp = *(inarray+i);
        *(inarray+i) = *(inarray+j);
        *(inarray+j) = tmp;

        }
    else
        {return j;   };

    };

}
```

Quick-sort in CLR pseudocode:

Algorithm 1 Quicksort main procedure.

```
1: QUICKSORT(A,p,r)
2: if p < r then
3:    q = PARTITION(A,p,r)
4:    QUICKSORT(A,p,q)
5:    QUICKSORT(A,q + 1,r)
6: end if
```

Algorithm 2 Partition procedure.

```
1: PARTITION(A,p,r)
2: x = A[p]
3: i = p-1
4: j = r+1
5: loop
6:    repeat
7:       j=j-1
8:    until A[j] ≤ x
9:    repeat
10:      i = i+1
11:   until A[i] ≥ x
12:   if i < j then
13:      exchange A[i] with A[j]
14:   else
15:      return j
16:   end if
17: end loop
```

Simulation of pointers via attributes. The language C has operators used to handle directly poiters, in particular & to denote addess-of-variable and $*$ to denote value of-variable-pointed-at. Let us see how linked lists are simulated instead in CLR pseudocode. A list is an object L and the pointer to its head is given by the attribute "head", so $head[L]$ is a pointer to the first element of L, such element is an object with three (or more) attributes:" prev" and "next" which are also pointers and" key" that holds the data item. So if x is (a pointer to) an element, $prev[x]$ is its predecessor, $next[x]$ is its successor in the list.
Interestingly, in line with ALGOL, the Algol-like pseudocode in [UAH74] does not seem to have a clean method for explicitly handling pointers. Instead a mechanism for parameter passing by "reference" is provided.

References

[CLR90] Thomas H. Cormen, Charles Eric Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* MIT Press/McGraw-Hill, Cambridge, Massachusetts, 1990.

[UAH74] J. D. Ullman, Alfred V. Aho, and John E. Hopcroft. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, 1974.