The `newalg` Package

Rick Farnbach*
Paul Furnanz†

August 27, 2002

Abstract

The package contains the definitions that are needed to typeset code algorithms in a pretty way. The Formatted algorithms follow the style set forth in the book "Introduction to Algorithms" by Corman, Leiserson and Rivest.

Contents

1 Introduction

The LaTeX macros which are described here allow descriptions of algorithms to be typeset in a pretty way. This is very useful for functional specifications for a software project or to document an algorithm for a white paper.

The idea for this macro package comes from the book "Introduction to Algorithms" by Cormen, Leiserson, and Rivest. Any examples in this document come directly from that book and should not be reproduced without proper attribution.

2 User Interface

2.1 The algorithm environment

`algorithm` Use the algorithm environment to typeset algorithm code. This environment makes

*Email: rick_farnbach@mentorg.com
†Email: paul_furnanz@mentrog.com

several new commands available that help in typesetting code algorithms. The algorithm environment uses math mode and the array enviroment to do the typesetting. Everything typed is interpreted in math mode. To leave math mode use the text command. Here is an example of the output produced by using the algorithm environment.

ALLOCATE-OBJECT()
1 **if** $free = \text{NIL}$
2 **then error** "out of space"
3 **else** $x \leftarrow free$
4 $free \leftarrow next[x]$
5 **return** x

```
\begin{algorithm}{Allocate-Object}{}
  \begin{IF}{free = \NIL}
    \ERROR{out of space}
  \ELSE
    x \= free \\
    free \= next[x] \\
    \RETURN x
  \end{IF}
\end{algorithm}
```

2.2 Flow Control Environments

IF Use the environment IF to format an if statement. When inside the IF environment, the ELSE macro becomes available to show the else clause. The environment takes one argument that is the condition for the if statement. For an example of its usage, see the above example.

FOR Use the FOR environment to format a for loop and takes one argument. There are two kinds of for loops supported by this macro. The first type of for loop is generally know as the *for-each* loop. This type of loop is used to iterate over the values of some set. The syntax for the argument to the environment is "`\EACH <var> \IN <set>`". The other type of loop supported is used to assign a variable to a range of values. The syntax for the argument in this case is "`<var> \= <beginning> \TO <end>`". Here is an example usage.

GREEDY-ACTIVITY-SELECTOR(s, f)
1 $n \leftarrow length[s]$
2 $A \leftarrow 1$
3 $j \leftarrow 1$
4 **for** $i \leftarrow 2$ **to** n
5 **do if** $s_i \geq f_j$
6 **then** $A \leftarrow A \cup i$
7 $j \leftarrow i$
8 **return** A

```
\begin{algorithm}
  {Greedy-Activity-Selector}{s,f}
  n \= length[s] \\
  A \= {1} \\
  j \= 1 \\
  \begin{FOR}{i \= 2 \TO n}
    \begin{IF}{s_i \geq f_j}
      A \= A \cup {i} \\
      j \= i
    \end{IF}
  \end{FOR} \\
  \RETURN A
\end{algorithm}
```

WHILE Uset the WHILE environment to format a while loop. The environment takes one argument. The argument is the exit condition for the loop. The loop will

iterate until the condition is false. Here is an example usage.

TREE-SUCCESSOR(x)
1 **if** $right[x] \neq$ NIL
2 **then return** TREE-MIN($right[x]$)
3 $y \leftarrow p[x]$
4 **while** $y \neq$ NIL and $x = right[y]$
5 **do** $x \leftarrow y$
6 $y \leftarrow p[y]$
7 **return** y

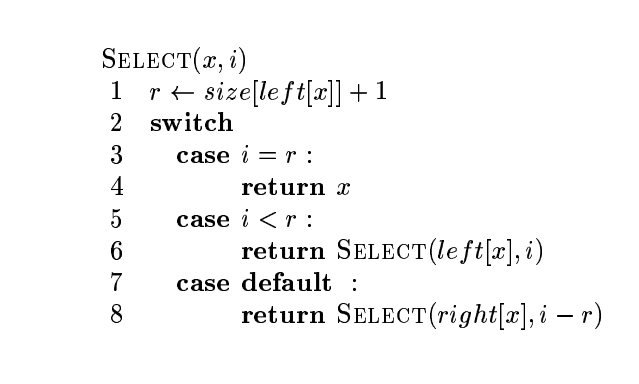
```
\begin{algorithm}{Tree-Successor}{x}
  \begin{IF}{right[x] \neq \NIL}
    \RETURN
       \CALL{Tree-Min}(right[x])
  \end{IF} \\
  y \= p[x] \\
  \begin{WHILE}
     {y \neq \NIL \text{and} x=right[y]}
    x \= y \\
    y \= p[y]
  \end{WHILE} \\
  \RETURN y
\end{algorithm}
```

REPEAT Use the REPEAT environment to format a repeat-until loop. The environment takes no arguments. The condition for the loop should be given after the `\end{REPEAT}` line. Here is an example usage.

HASH-SEARCH(T, k)
1 $i \leftarrow 0$
2 **repeat**
3 $\leftarrow h(k, i)$
4 **if** $T[j] = k$
5 **then return** j
6 $i \leftarrow i + 1$
7 **until** $T[j] =$ NIL or $i = m$
8 **return** NIL

```
\begin{algorithm}{Hash-Search}{T,k}
  i \= 0 \\
  \begin{REPEAT}
    j \= h(k,i) \\
    \begin{IF}{T[j] = k}
      \RETURN j
    \end{IF} \\
    i \= i+1
  \end{REPEAT} T[j]=\NIL\text{or} i=m \\
  \RETURN \NIL
\end{algorithm}
```

SWITCH Use the SWICH environment to format a very general switch statement. The environment is like an itemize environment. Use the command `\item{<condition>}` to create a new case label. The conditions can be anything, and don't all have to test the same variable. This is the most general switch statement. The formatting conventions show that the first case condition to match will be executed. The text `\DEFAULT` may be used for the condition to provide a default action. Here is an example usage.



```
SELECT(x, i)
1  r ← size[left[x]] + 1
2  switch
3    case i = r :
4         return x
5    case i < r :
6         return SELECT(left[x], i)
7    case default :
8         return SELECT(right[x], i − r)
```

```
\begin{algorithm}{Select}(x, i)
  r \= size[left[x]] + 1 \\
  \begin{SWITCH}
  \item{i = r} \\
    \RETURN x
  \item{i < r} \\
    \RETURN
    \CALL{Select}(left[x], i)
  \item{\DEFAULT} \\
    \RETURN
    \CALL{Select}(right[x], i - r)
  \end{SWITCH}
\end{algorithm}
```

2.3 Macros

`CALL` Use the CALL macro to format a function call. The macro takes one argument. The argument is the name of the function to call. It is usually followed by the parametes to the function call. For example, "`\CALL{Sort-Array}(array, length)`".

`ERROR` Use the CALL macro to signal that some sort of error has occured. The macro takes one argument that the reason for the error. The text will be formatted in text mode (not math mode) and will be surrounded by quotation marks.

`algkey` Use the algkey to format key words. If this package does not define a keyword that you want to use, then this macro is used to format your keyword like the other keywords.

2.4 Additional keywords and symbols

`RETURN` Use the RETURN macro to print out the return keyword. Usually this macro is followed by information that is to be returned from the algorithm but this is not an argument to the macro.

`NIL` Use the NIL macro to print the nil keyword. This keywod is used to represent

`\=` a variable that has no value assinged. Use the text `\=` to signal assinment. This command produces this symbol in the formatted text, "←".

3 Future Work

The current implementation of the algorithm environment is sensitive to the proper placement of \\ in the text. See the examples for this. The environments should work without being so fussy on this point. (Sometime you need a \\ at the end of and environment, sometimes you don't).

I would like the syntax of the repeat loop to be the same as the while loop. I was having some trouble getting the stack commands to work, so that I could save of the argument. This environment is not very consistent with the rest of the algorithm environments.

There is probably a better way to do the formatting then using the array environment. Currently LaTeX is formatting the algorithms by using the array environment. This is pretty silly, because this is not really an array.

You cannot center the algorithm environment. This is probably becase it is being implemented as an array. The current workaround for this problem is to include the algorithm in a `\begin{minipage}{1pt}` ... `\end{minipage}`. Seems to work in every case that I have come accross.

There is probably a better way to make a mode that is like math mode that does not insert `$` characters everywhere.

I am not very experienced in writing modes for LaTeX, so if you have any suggestions for improvements or know how to solve any of the above listed problems, please send me email. The address is on the front page of this document.