# Heapsort

In this exercise you will use a subroutine which uses the "heapsort" algorithm to sort a list of numbers read from a file, and you will learn how to assemble programs from pieces compiled from separate files. You will also get an introduction to two useful programming tools, the Revision Control System (RCS) and the `make` program.

## 1. Problem

The "heapsort" is one of the best ways to sort data, because the time it takes to sort a list of $N$ numbers grows only like $O(N \log_2(N))$, not $O(N^2)$, and it is relatively easy to program (although it is more complicated than the bubble sort or the selection sort).

Instead of writing a heapsort routine, you will use one someone else has written. You will find a description of the heapsort algorithm as well as a "pseudo-code" version of a subroutine to sort an array of numbers using that algorithm in the well known book *Numerical Recipes (The Art of Scientific Computing)* by W.H. Press, B.P. Flannery, S.A. Teukolsky and W.T. Vetterling (Cambridge University Press, 1986). You should type the subroutine into the file `heapsort.f`. It is not very long, but it is written in a style that is not quite official Fortran, so you will have to do some conversion as you type, such as providing line numbers for `DO` loops and the like.

You should then write a main program (or modify a copy of Exercise 12) which behaves exactly like Exercise 12, except that it calls the subroutine `HEAPSORT` instead of the routine `SORTEM`. The first argument to `HEAPSORT` will still be the number of items to sort, and the second argument will still be the name of the array containing the REAL data to sort.

Your program should *not* include the `HEAPSORT` routine in the same file as your main program. Instead, compile `heapsort.f` by itself with the command

```
%   f77 -c heapsort.f
```

to produce the file `heapsort.o`. (The `-c` flag means "compile only", and the "%" represents the Unix prompt.) Then do the same thing to your own program, which does not have heapsort in it. That is, compile it with "f77 -c", as in:

```
%  f77 -c mavassar13.f
```

You will then have two separate files with names which both end in ".o". These are "*object*" files, and they contain the translation of your Fortran program into machine code, but they still must be "linked" (or "loaded") together to form an executable file. You can *link* (or "*load*") these separate modules to create the executable file with a command like the following:

```
%  f77  mavassar13.o  heapsort.o  -o mavassar13
```

The Fortran compiler will recognize that these files have already been compiled and will simply load them to create the executable file.

## 2. Input/Output

Just as in Exercises 11 and 12 you should read the numbers to be sorted from a file. As in Exercise 12, the numbers can be assumed to be real, and the sorted list should be written to an output file, not to the terminal. The output file should have the same base name as the assignment, using your userid and the exercise number, and the filename extension `.rpt` (Example: `mavassar13.rpt`). You should print a comment to the user telling them where the sorted data have been put, and the number of items that have been sorted.

## 3. Testing

To test and demonstrate your program run it using the 2000 real numbers in the file `R2000.d` in my directory `~myers/fortran` with the Unix time command and note the time taken. Run your selection sort program (Exercise 12) on the same data file and note the time taken. Which one is faster?

If you like, you can also try timing your bubble sort program (Exercise 11) using the same data file. This is easily done using the Unix "input redirection" operator "<" to tell the program to read input from a file not the terminal. The command would be something like:

```
%  mavassar11 < R2000.d
```

Which of the three algorithms is the slowest? Which is the fastest?

To turn in your assignment copy *only* your main program file to my inbox directory. I already have my own copy of `HEAPSORT`, which I will use to test your programs. Your sample run should show your run of both the heapsort and selection sort programs, with the times taken by each.

If you copy your program from Exercise 12 be sure to remove `SORTEM` completely!

## 4. Using RCS

To gain experience with using RCS (the Revision Control System) you should also include at the top of your program (after your name and date and the like) an RCS comment line like the following:

```
C  @(#) $Revision: 1.3 $ :  $Date: 2003/03/15 12:35 $ :  $Author: mavassar
$
```

To get the actual date, time and an updated version number into this comment line you should "check in" the file to RCS, using the `ci` command, like so:

```
%  ci mavassar13.f
```

Once you have done this you will see that your file `mavassar13.f` is no longer in the current directory, but that a new file, called `mavassar13.f,v` now exists. This is the "version" file, which contains the latest version of the file and revision history. You can retrieve an up-to-date copy of your file from the version file by "checking out" the file from RCS with the `co` command, like so:

```
%  co mavassar13.f
```

Your "working file" will reappear in the directory, and the version file will also be there. If you look at your working file now you will see that the revision number, date and author have all been updated.

To view the revision log for the file give the command

```
%  rlog mavassar13.f
```

From now on, if you make changes to your program, you can check them into RCS using `ci`, and the changes will be stored in the version file. You can then check out either the latest version or any previous version. You can also use the `rcsdiff` command to look at the differences between different versions of a file.

## 5. The `make` program

Another useful programming tool in Unix is the `make` program, a utility which manages compilation and can assemble programs from a number of different files. All you need to do is create a file containing simple instructions telling `make` what to make (called a "target"), what to make it from (called the "dependencies"), and how to make it (called a "rule").

The best way to learn to use `make` is by an example. If for this exercise your "target" is the program `mavassar13`, which "depends" upon both `mavassar13.o` and `heapsort.o`, then the rule for compiling them is to simply give the f77 command with both file names.

To have `make` do this for you you create a file called `Makefile` in the current directory containing the following lines:

```
mavassar13:  mavassar13.o heapsort.o
⟨tab⟩        f77 mavassar13.o heapsort.o -o mavassar13
```

On the first line, the target `mavassar13` is followed by a colon, and then the list of files which are needed to build the target. On the next line, which must begin with a ⟨*tab*⟩, is the rule for making the target. To have `make` build the target you then simply give the `make` command and name the target, like so:

```
%  make mavassar13
```

The `make` program knows that "`.o`" files are to be made from "`.f`" files and will compile them with `f77 -c` (or you can specify your own rule). The `make` program will also look at the date and time on each file to see which file is out of date, and will only re-compile those parts which have changed. Thus if you change your program but don't change the heapsort routine, `make` will only re-compile your program and then invoke the rule to build the executable. Try it. You can have many rules for many targets in a Makefile.

## 6. Reading

You should read at least the background material describing the heapsort algorithm in *Numerical Recipes (The Art of Scientific Computing)* by W.H. Press, *et al.*. This book also contains a collection of other useful subroutines for a wide variety of scientific and engineering applications.

You can learn more about sorting in general from *Sorting and Searching* (Volume 3 of *The Art of Computer Programming*), by Donald Knuth (Addison Wesley, Inc., 1973).

For more information on RCS and `make` see the man pages for `rcs`, `ci`, `co`, `rlog`, `rcsdiff`, and `make`, and the chapters on RCS and `make` in *Unix for FORTRAN Programmers*, by Mike Loukides (O'Reilly & Associates, Sebastopol, Calif., 1990). A brief tutorial is given in the handout "*A Quick Introduction to RCS (the Revision Control System)*" distributed with this exercise.

For more about Unix input/output redirection you can read the on-line manual page for `tcsh` by giving the command '`man tcsh`', or see any introductory book on Unix.