CMSC 202	**Project 1**	Spring 1996
Sections 101, 102 & 103	**due: Tuesday, 12 March**	Handout #12

Objectives

This assignment has two main objectives:

1. to give you practice building and manipulating linked structures; and
2. to surreptitiously introduce you to the concepts that underlie data abstraction.

Background

Spring is here and the Clintons have decided to clean up their storerooms. They have agreed to collect the items they find into files, to collect items and files into superfiles, to collect items, files and superfiles into superduperfiles, *etc.* The items in each collection will be numbered, starting from zero (the President's nickname around the White House is after all "Bill C.").

The Clintons decide they will begin their spring cleaning at the OMB. Here they find a wide variety of integers and real numbers that need to be organized. Unfortunately, when they try to use arrays to hold the items to be organized, the compiler complains—something about incompatible types. Slowly it dawns on them that an array in C can hold only a single type of object, but they want to build arrays that store several different types of objects (some of which are themselves arrays). In desperation, the first couple turns to you.

Assignment

You are to restore the Clintons' dreams of organizing their records by implementing a new type of object called a *multi–array.* A multi–array is like an array, in that it can store a fixed number of objects, which are labeled by numbers starting from zero. Unlike a normal array though, a multi–array can hold elements of three different types: integers, real numbers, and other multi–arrays.

Figure 1 shows a picture of how multi–arrays can be organized. A multi–array is a pointer to a structure of two fields. The first field holds the number of elements in the multi–array. The second field is a pointer into an ordinary C array of structures.[1] Each of these structures holds a type (one of `EMPTY`, `INTEGER`, `REAL`, or `ARRAY`), and a pointer representing the object stored. Notice that since this latter field must be a pointer, integers and reals are stored not directly as `int`s and `float`s, but rather as pointers to `int`s and `float`s. Because a multi–array is already a pointer, this extra level of indirection is not needed for multi–array elements that are themselves multi–arrays.

[1] Figure 1 shows this as a pointer to the zeroth element of the array. If you prefer, you may make this be a pointer to the entire array.

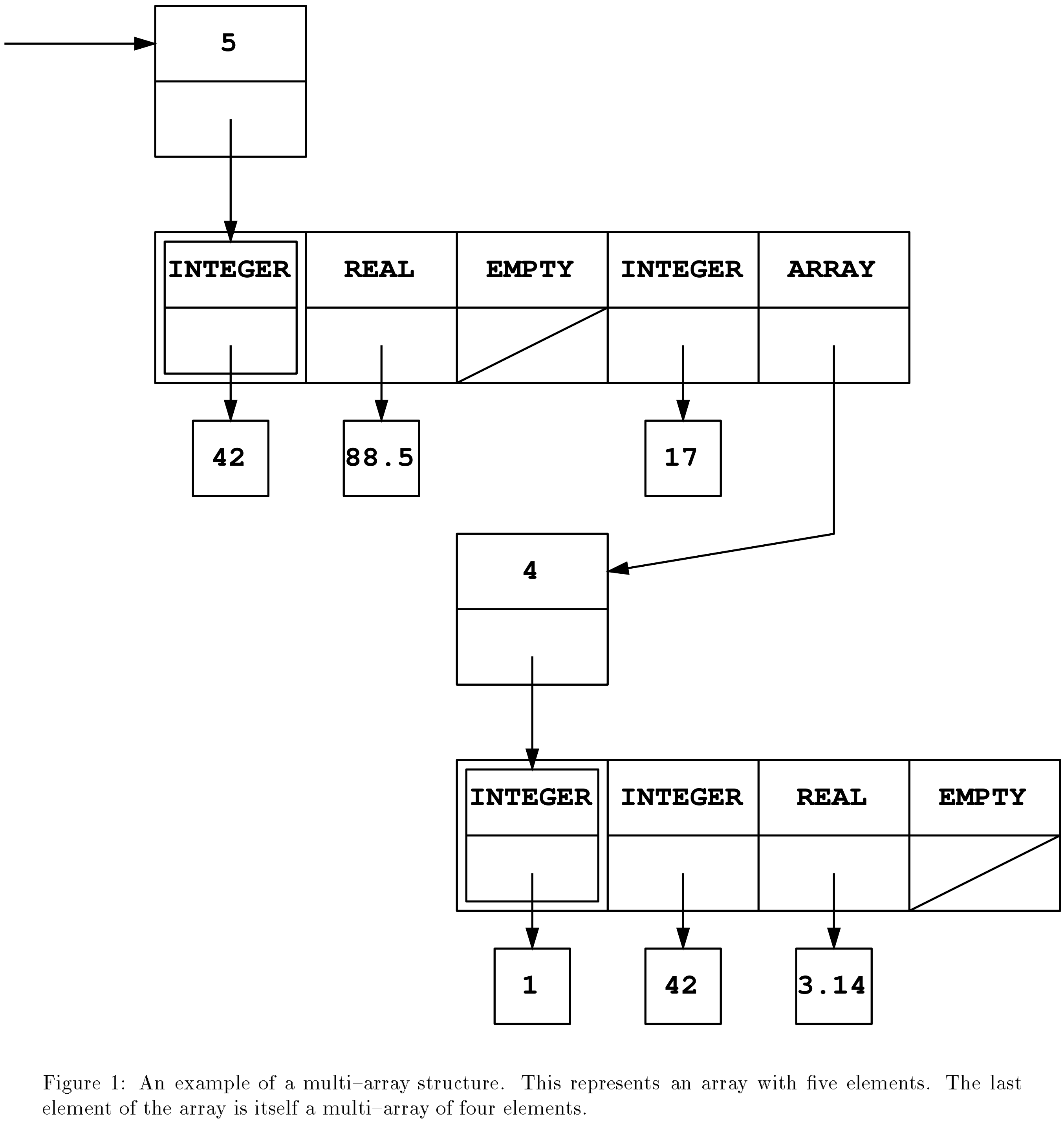

Figure 1: An example of a multi–array structure. This represents an array with five elements. The last element of the array is itself a multi–array of four elements.

Once you have implemented multi–arrays, you will build a program that:

- Reads a description of a multi–array from a file, and builds the corresponding multi–array.
- Prints the multi–array it read to the screen.
- Repeatedly asks the user for a number (either an integer or a real number). For each number, the program searches the multi–array it read for instances of that number. Whenever it finds the desired number in the multi–array, it prints the sequence of subscripts that leads to the number. For example, if the user wanted to find occurrences of the integer 42 in the multi–array shown in Figure 1, the program would print `[0]` and `[4][1]`, identifying the two occurrences of 42 in that multi–array.

Here is a sample trace of how your program should look in action. User input is shown in *italics*:

```
umbc9[101]  proj1
Please enter the name of the multi-array description file:  myarray.dat
The multi array is:
{42; 88.5; UNSET; 17; {1; 42; 3.14; UNSET}}
What number do you seek?  integer 42
42 was found in the following positions:
  [0]
  [4][1]
What number do you seek?  real 3.14
3.14 was found in the following positions:
  [4][2]
What number do you seek?  real 2.71828
2.71828 was found in the following positions:
  2.71828 was not found.
What number do you seek?  real 42.0
42.0 was found in the following positions:
  42.0 was not found.
What number do you seek?  rational 3/4
'rational' is not a known number type.
What number do you seek?  integer seven
'seven' is not a legal integer.
What number do you seek?  ^D
Bye!
umbc9[102]
```

Input File

The input file will contain a single multi–array specification. A multi–array is specified by a line containing the word `array`, followed by a line containing the number of items in the array, followed by that number of object specifications. An integer is specified by a line containing the word `integer` followed by a line containing the integer. A real number is specified by a line containing the word `real`, followed by a line containing the real number. An array element that is unset is specified by a single line containing the word `unset`.

Figure 2 shows the input file that would be used to specify the multi–array shown in Figure 1. You may assume that the input file is well–formed; you need not perform any error checking on its contents.

```
array
5
integer
42
real
88.5
unset
integer
17
array
4
integer
1
integer
42
real
3.14
unset
```

Figure 2: The input file that specifies the multi–array shown in Figure 1.

Routines for Manipulating Multi–arrays

In programming this project, you must clearly separate the code that implements multi–arrays from the code that uses them. The code that implements multi–arrays must reside in its own `.c` file. Routines that you will need to include in that file include:

- A routine that takes an integer as its argument and builds a new multi–array that can hold that number of elements. Your routine should return `NULL` if it unable to allocate all of the memory required.
- A routine that makes a copy of an existing multi–array. The entire structure must be duplicated, including sub–arrays. That is, the old structure and the new structure must not have any pointers that refer to the same space.
- A routine to set a particular element of a multi–array. In addition to the multi–array and index, this function must take a pointer and a type as arguments. The type must be one of `INTEGER`, `REAL`, `ARRAY`, or `UNSET`. Since the pointer might be a pointer to any of three types of objects, its type must be specified as `void *`. The object that is actually stored must be a *copy* of the object handed to the function (regardless of whether the object is a pointer to an integer, a pointer to a real, or another multi–array).
- A routine to retrieve an element from a multi–array. Again, the return type should be `void *`; it is up to the caller to properly interpret the value returned.
- A routine to retrieve the type of an element in a multi–array (which will be one of `INTEGER`, `REAL`, `ARRAY`, or `UNSET`).
- A function to test whether a given index is legal subscript for a given multi–array; it should return `TRUE` if the index is within the appropriate bounds, and `FALSE` otherwise.
- A routine that returns the size of a given multi–array.
- A routine that prints a given multi–array. To implement this function you will want to set up a mutual recursion with another function that prints out a single multi–array element.

All manipulation of multi–arrays in the rest of your program must be accomplished by calling these routines.

Submitting the Project

You must submit the following files:

1. the file containing your main function;
2. all your interface files;
3. all your implementation files; and
4. your `Makefile`. You must submit a `Makefile` that causes correct compilation of your source files upon issue of the command '`make`'. The executable resulting from the compilation must be named `proj1`.

To submit the files, use the `submit` program. For example, to submit the files `proj1.c` and `proj1_aux.c`, enter '`submit cs2021 proj1 proj1.c proj1_aux.c`'.

You can check your submission by entering '`submitls cs2021 proj1`'.

The homework is due on Tuesday, 12 March. Submissions received by midnight on the due date will receive the on–time bonus. There is an automatic extension of two days, so submissions received by midnight of 14 March will receive full credit. No submission will be accepted after 14 March. Don't forget to include a comment at the top of your program stating your sources of help (as described in the Prospectus).