NAME

allocateArray, allocateArrayV, deallocateArray, deallocateArrayV − create and destroy arrays with an arbitrary number of dimensions

SYNOPSIS

#include <dynArray.h>

void* allocateArray(int *datasize***, int** *ndim*,**...);**
void* allocateArrayV(int *datasize***, int** *ndim***, int** *dim[]***);**
void deallocateArray(void* *ptr***, int** *ndim*,**...);**
void deallocateArrayV(void* *ptr***, int** *ndim***, int** *dim[]***);**

DESCRIPTION

These functions create and destroy multidimensional arrays whose dimensions are not known until runtime so that can be passed as arguments to functions without the dimensions being known at compile time.

The functions **allocateArray**() and **allocateArrayV**() each allocate memory for an array with elements that require *datasize* bytes and have *ndim* dimensions. The calling program is required to cast the return value to the appropriate pointer. **deallocateArray**() and **deallocateArrayV**() free memory pointed to by *ptr* that was previously obtained by **allocateArray**() or **allocateArrayV**().

The functions **allocateArray**() and **deallocateArray**() each require a total of *ndim*+2 arguments where *ndim* is the number of dimensions of the array. The last *ndim* arguments are the actual dimension values. Alternatively, the last argument of **allocateArrayV**() and **deallocateArrayV**() is an *ndim* element vector (1-D array) that contains the array dimension values.

The memory allocated for the data portion of the array is allocated at one time and so will be contiguous. This means that offsets to "slices" in the array are possible, just as if the array had been statically allocated.

The value of *datasize* must be the size in bytes of the basic datatype of the array. Calling programs may use the **sizeof**() operator or constants that are predefined in dynArray.h: **SZbool**, **SZchar**, **SZshort**, **SZint**, **SZlong**, **SZuchar**, **SZushort**, **SZuint**, **SZulong**, **SZfloat**, **SZdouble**, and **SZldouble**.

RETURN VALUE

The allocation functions each return a pointer that must be cast to a pointer of appropriate type and level of indirection (dimension). The deallocation functions do not return any value.

EXAMPLE

A 10 by 15 two-dimensional array of integers could be allocated with

```
int** a = (int**) allocateArray(SZint, 2, 10, 15);
```

or

```
int dim[2] = {10, 15};
int** a = (int**) allocateArrayV(SZint, 2, dim);
```

In both cases the return value from the function has been cast to a pointer to a two-dimensional array of integers. The resulting array can be accessed as if it had been declared as

```
int a[10][15];
```

but it can also be passed to and used by functions without knowing its dimensions at compile-time. A three-dimensional array of floating point numbers with dimensions 8, 5, and 13 can be allocated with

```
float*** b = (float***) allocateArray(SZfloat, 3, 8, 5, 13);
```

or

```
int dim[3] = {8, 5, 13};
float*** b = (float***) allocateArrayV(SZfloat, 3, dim);
```

Elements in the array can be accessed with references like b[0][2][5]. Dynamic array memory is be

released with commands such as

```
deallocateArray(a, 2, 10, 15);
```

and

```
int dim[3] = {8, 5, 13};
deallocateArrayV(b, 3, dim);
```

Notice that the same dimensions used to allocate the array must be used when it is deallocated.

AUTHOR